Politecnico di Milano
Department of Aerospace Science and Technology
Doctoral Programme in Aerospace Engineering

AI-based Guidance for Spacecraft Proximity Operations around Uncooperative Targets

Doctoral Dissertation of:
Andrea Brandonisio

Supervisor:
Prof. Michèle Lavagna

Tutor:
Prof. Alberto Matteo Attilio Guardone

Coordinator:
Prof. Pierangelo Masarati

Year 2024 – Cycle XXXVI

Are we out of the woods?

Part of this Ph.D. has been funded by Fondazione Fratelli Confalonieri, Milan, Italy.
https://www.fondazionefratelliconfalonieri.it

Abstract

In recent years, space research has heavily shifted its focus towards enhanced on-board autonomy for spacecrafts' on-orbit servicing (OOS), which, together with the automatization of proximity operations, may include a great variety of activities aimed to increase the reliability, flexibility, and cost-effectiveness of a space mission, lowering down all the possible associated risks. Concurrently, the rapid development of Artificial Intelligence (AI) is strongly influencing aerospace research, particularly in the framework of autonomous guidance, navigation and control (GNC) systems. Therefore, among all, these two research branches may converge towards the autonomy and adaptability of GNC, which may represent one of the most effective ways of reducing human intervention during spacecraft missions. Within this context, adaptive guidance depends on the ability of the system to build a map of the uncertain environment, figuring out its location inside of it and accordingly determining the control law. Thus, this autonomous navigation problem is usually framed as an active Simultaneous Localization and Mapping (SLAM) problem and modeled as a Partially Observable Markov Decision Process (POMDP). Nowadays, the state-of-the-art methodology to approach POMDP optimization is Deep Reinforcement Learning (DRL), which relies on the Proximal Policy Optimization (PPO) algorithm for continuous state-action space models.

In this context, this present thesis wants to investigate the design and development of an autonomous agent capable of cleverly planning trajectory around unknown and uncooperative target objects to optimize the shape reconstruction of the target itself.

The first part of the work covers all the background analysis and mathematical models selection in order to evaluate the best options for the environment and agent design. These choices are afterwards explained and tested to assess the feasibility of the approach and the baseline performance results. The guidance algorithm is evaluated in terms of target map reconstruction, by rendering the space object with a triangular mesh and then considering the number of quality images for each face. A major differentiation in the algorithm implementation is provided by the employment of either a discrete or a continuous action space, a perfect or noisy state space, and, moreover, different neural network architectures. The proposed model is trained and then extensively tested, always starting from random initial conditions, to verify the generalizing capabilities of the DRL agent.

Once the foundation of the agent is set and the robustness of the approach is guaranteed, this guidance block is inserted into a more general AI-based GNC system, based on image processing and navigation tools. Extensive tests, in different scenarios, are performed to verify the capability of the agent and the overall pipeline. In the end, the applicability of DRL methods and neural networks to support autonomous guidance is further corroborated with other kinds of guidance and control environments, aiming to assess the training and testing methodology.

Overall, the proposed approach for autonomous guidance design is demonstrated to be an effective possibility towards an increased autonomy of the spacecraft's GNC system.

Table of Contents

List of Figures

List of Tables

List of Acronyms

2BP Two-body problem

A2C Advantage Actor-Critic

AI Artificial Intelligence

ANN Artificial Neural Network

CAM Camera

CHS Chaser

CM center of mass

CNN Convolutional Neural Network

CW Clohessy-Wiltshire Model

DOF Degrees of Freedom

DRL Deep Reinforcement Learning

DQN Deep Q-Network

ECI Earth-centerd Inertial

EKF Extended Kalman Filter

FFNN Feedforward Neural Networks

FoV FoVField of View

GNC Guidance, Navigation & Control

HW hardware

IG Image Generation

IP Image Processing

LEO Low Earth Orbit

LSTM Long-Short Term Memory

LVLH Local Verical, Local Horizontal

MDP Markov Decision Process

MIB minimum impulse bit

ML Machine Learning

MLP Multi-Layer Perceptron

MPC Model Predictive Control

MSE Mean Squared Error

NAV Navigation

NN Neural Networks

OBC On-Board Computer

OD Orbit Determination

OOS On-Orbit Servicing

PD Proportional-Derivative

Ph.D. Philosophiae Doctor

PIL Processor-In-the-Loop

POMDP Partially Observable Markov Decision Process

PPO Proximal Policy Optimization

RL Reinforcement Learning

RMSE Root Mean Squared Error

RNN Recurrent Neural Network

SD Standard Deviation

SLAM Simultaneous Localization and Mapping

SOTA State-of-the-Art

SPC Stereophotoclinometry

SW software

TD Time Difference

TIR Thermal Infrared

TL Time Learning

TRG Target

TRPO Trust Region Policy Optimization

VIS Visible Imaging Sensor

CHAPTER 1

Introduction

> Trying to learn to walk like heroes we thought we had to be
> Well, after all this time, to find we're just like all the rest
>
> — Bruce Springsteen, *Backstreets*

Nowadays, enhanced autonomy is the research driver of most of the leading space agencies, as spacecraft independence would allow for reliable, cost-effective, lower-risk services, and for much more flexibility in mission planning. Moreover, in the fast-developing field of space exploration and satellite deployment, the concept of On-Orbit Servicing (OOS) has emerged as one of the primary solutions in order to extend the lifespan of a mission, to enhance the capabilities and ensure the sustainability of spacecraft, especially orbiting around the Earth. This means that OOS can include a wide selection of activities of great interest: maintenance, refuelling, debris mitigation, upgrade, repair, assembly, relocation, orbit modification and non-contact support are some of the operations which can be included in the definition [1, 2]. A high level of autonomy while carrying out these activities would lead to greater

benefits. That is why, at the heart of this groundbreaking field lies a crucial aspect: the spacecraft's guidance and control autonomy. Indeed, any spacecraft with on-orbit servicing objectives may rely on a sufficient autonomy level, enabling them to execute intricate manoeuvrers or tasks without constant human intervention. Therefore, autonomy is a benchmark on which ensuring precise navigation, rendezvous, docking, or the execution of maintenance operations, which require cutting-edge technologies, algorithms, and sensors that increase the capability of these vehicles to adapt, respond, and perform delicate operations with unparalleled precision in a challenging environment as the space one. Guidance and control play an integral role in this autonomy level: the first requires the use of advanced algorithms to determine optimal trajectories in very different types of space scenarios exploiting the information coming from integrated systems made up of a multitude of sensors, i.e. as star trackers, GPS, and LiDAR, that enable precise positioning and orientation, facilitating safe proximity operations and challenging manoeuvrers around other space bodies or objects. Control autonomy, instead, grants spacecraft the ability to make real-time decisions and adjustments, ensuring stability, collision avoidance, and successful execution of complex tasks. This autonomous control may be strongly empowered by AI-driven software, adaptive control algorithms, and fault-tolerant systems that enable spacecraft to respond promptly to unforeseen situations, guaranteeing mission success while minimizing any possible risks. Subsequently in the constant evolution of on-orbit servicing, the focus on spacecraft guidance and control autonomy emerges as a fundamental aspect which strongly drives the success and efficiency of missions, in order to make the way to a new era of space exploration and applications where spacecrafts can navigate and operate with unprecedented precision and adaptability.

Most of the times, this growing hunger for autonomy can be only satisfied by the *brand-new* player that stole the scene in these recent decades: artificial intelligence and machine learning. Indeed, in the last few years, the number of works on machine learning techniques applied to space-related problems is continuously growing, especially in the context of guidance, navigation and control [3, 4, 5], affecting almost all the most important aspects of the GNC in a great variety of scenarios from feasibility studies to on-board applications [6, 7]. The potential of machine learning, mainly resulting from its adaptability and diversity, allows it to be tailored for every need, offering different solutions (supervised, unsupervised and reinforcement learning) and declinations (i.e. linear, convolutional, recurrent neural networks, etc.). This enables ML applications to span from Convolutional Neural Network (CNN) based algorithms exploited to solve for image-based lunar landing navigation [8] or spacecraft rendezvous [9], through radial-basis networks helping for relative navigation [10] and predictive control [11, 12]. Focusing, instead, on Reinforcement Learning (RL) as a sub-discipline of machine learning, the peculiarity of having

agents capable to learn how solving tasks (like navigation or planning) within a potentially changing environment, ensures the power of adapting the mathematical model to a practically endless number of space-related guidance and control applications. Therefore, among all, scenarios involving spacecrafts hovering or orbiting around irregular small bodies, such as asteroids and comets, have been studied in [13, 14, 15]. Here the spacecraft's guidance design problem, aimed to plan the chaser trajectory based on small-bodies imaging, is modelled as Partially Observable Markov Decision Process (POMDP), adopting an active SLAM formulation based on RL algorithms, such as NFQ (Neural Fitted Q) and DQN (Deep Q Network), which nowadays have been outperformed by other methods more suitable for continuous state-action space problems, like PPO [16]. In this research field, a major contribution has been given by [17, 18, 19], in which, among others, planetary landing and close proximity operations have been investigated. This particular approach has been proven effective in different environments as the circular restricted three-body problem (CR3BP) framework [20] and also compared to cloning techniques, as done by [21].

This brief overview of space applications demonstrates how ML is certainly suitable for helping OOS operations. They usually share similar scenarios, in which two objects are in relative motion and interact with each other in different levels, generally with the first one, referred to as the *servicer*, performing some actions on the second, called *client*. Then, the Ph.D. thesis here presented aims to study the development of an innovative adaptive guidance and control algorithm for the path-planning of a chaser spacecraft trajectory around an artificial object, designed to map and reconstruct the shape of the target. This client object may be unknown and uncooperative so it does not interact directly with the servicer, and it is not able to exchange information with it. In this context, the spacecraft shall autonomously explore the surrounding environment, understanding where it is located inside of it, and consequently choosing the actions to take, according to a certain objective function to which the spacecraft is assigned. This kind of problem falls in the domain of active Simultaneous Localization and Mapping (SLAM) framework, which is a problem formulation that has been widely researched in the last few years, particularly in the field of robotics [22, 23]. Any active SLAM problems may be phrased as an already introduced POMDP, which derives from the Markov Decision Process (MDP), and entails an autonomous agent that interacts with the environment and exchanges information with it [24]. According to [25], in the current state-of-the-art, DRL is the most common method used to solve both MDP and POMDP, aiming to optimise a decision-making policy of the learning agent. RL algorithms are a powerful tool when dealing with decision-making problems and their perfect compatibility with Artificial Neural Network (ANN) as function approximators allows them to improve the generalizing capabilities of the resulting policy, and to solve more and more

complex problems characterized by high-dimensionality and continuous state and action spaces [24]. Even if POMDP represents the first step in applications of AI policy models in real-world scenarios, it is worth underlining how it still remains problematic when dealing with partial observations as full states, thus violating the Markov property of state transitions: this is particularly challenging when the system is strongly affected by historical information or large state/action spaces.

In conclusion, this work takes shape from [26], where, for the first time, these tools were exploited for the guidance and control of a chaser spacecraft aimed at reconstructing the shape of an artificial object. In the following sections the research objectives, the overview of the work and the bibliographic disclaimer will be presented in Section 1.1, Section 1.2 and Section 1.3 respectively.

1.1 Research objectives

In the previously presented context, this thesis' work humbly wants to study potential advancements in the Guidance, Navigation & Control (GNC) design to move steps forward in the definition of spacecraft's guidance and control techniques characterized by an increased level of autonomy. In particular, the following high-level question is posed.

> How exploitable are the DRL techniques for spacecraft path-planning optimization during proximity operations around uncooperative and unknown space objects?

This initial research objective is thus partitioned and organized into different activities aimed at assessing the feasibility of the methodology and understanding the potentiality of the performances. In particular, the answer is constructed starting from the development of the algorithm baseline model, identified as PPO, and the selection of the model's architecture among all the possibilities offered by the neural network framework; in particular, the research will be restricted to linear networks' models and recurrent models. Afterwards, the algorithmic framework will be analysed especially in terms of reward shaping and environmental initial conditions influence. These two aspects are believed to be highly relevant in the assessment of the performance of a DRL-based agent since they define the objective of the policy to be optimized and constrain the environment in specific scenarios.

The automatic implication of the answers to the primary objective is embodied in the second research question, which focuses on the robustness and flexibility of the adopted approach. Both of them are typical machine learning characteristics, which are increasingly difficult to reach when the problem becomes more complex and realistic.

> Could an AI agent - devoted to design spacecraft guidance in scenarios affected by uncertainty - be robust, get to the mission objectives, and be as performant as a classical approach?

With this second objective, the intention of guiding the methodology into the definition of a pseudo-real architecture is outlined. In particular, the thesis wants to focus its attention on two of the most important aspects of every DRL applications: the action and the state space models. Both of them are analysed in order to achieve the control modulation and the robustness of the agent to input errors or uncertainties. For doing this, different model architectures can be investigated to counteract the width of the new action-state space. In the end, in order to prove the effectiveness of the proposed method, also a benchmark analysis turns out to be necessary for the good quality of the results.

Once also this second question is answered, the natural consequence leads to the analysis of the policy agent's behaviour when placed in a complete GNC pipeline. This brings the third research question.

> How does the AI agent perform whenever part of a full AI-based GNC pipeline (IG+IP+NAV+G&C) for autonomous relative dynamics management?

This request raises the necessity of having access to GNC tools, whose definition and design go beyond the aims of this work. Nevertheless, image-based processing and navigation methodologies, coupled with the relative image generation tool, have been selected to be compatible and reliable for a full AI and image-based GNC algorithm. The integration of these tools makes even more realistic the study of the adopted guidance and control model, which, now, can be tested and analysed in an uncomfortable environment, which may diverge with respect to the experienced one.

In the end, after having deeply analysed and characterized the proposed techniques both in terms of study methodology, analysis architecture and model performance, one last obvious question arises.

> Could a proper framework of uncertainty and complexity be defined to develop a robust AI-based guidance, still reliably applicable to very different space scenarios?

Therefore, the thesis also tries to answer this last research question, investigating the same methodology taken at different stages of complexity applied to different scenarios, that still are characterized by the same degree of complexity.

In particular, this objective is pursued by analysing three further situations: planning of debris avoidance strategy, guidance and control of launcher first stage planetary landing and robotic arm guidance and control for uncooperative target grasping. All these scenarios have been evaluated with the same methodology design.

All these topics are fully addressed in the following chapters.

1.2 Thesis Overview

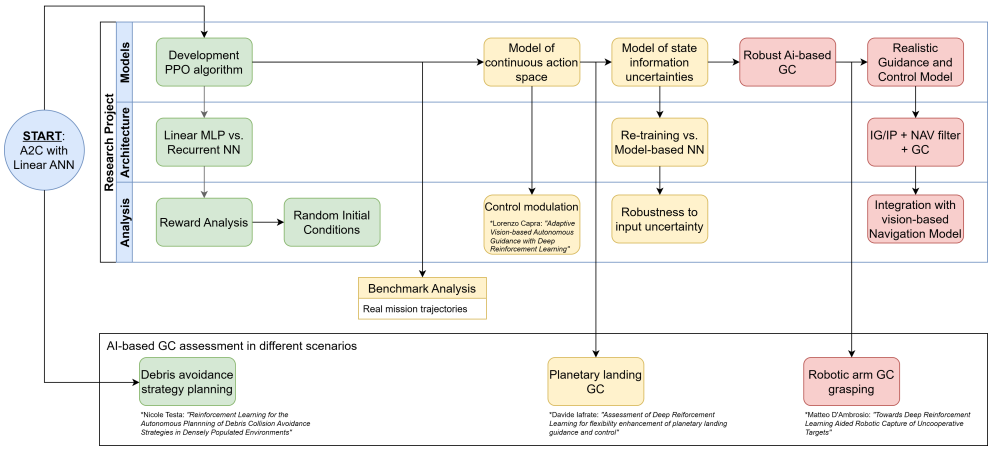


Figure 1.1: Schematic of the work done in this thesis.

The overall thesis can be subdivided into three main blocks, as highlighted in different colours in Fig. 1.1. Each of the blocks is mainly linked to the research questions described in the previous section. The first part, depicted in green, is related to the analysis of the feasibility of DRL for spacecraft path-planning in terms of network architecture models, reward shaping and initial conditions robustness. Following the background and state-of-the-art analysis described in Chapter 2, this topic will be treated in Chapter 3, Section 4.1 and Section 4.2. The second block, highlighted in yellow, wants to answer the second thesis question with a new definition and analysis of environment state and action spaces, correlated to a benchmark analysis aimed to verify the methodology this thesis wants to propose. This part will be described from Section 4.3 to Section 4.6. The third and last block, in red, answers the third objective questions on the performance and behaviour analysis of an entire AI-based GNC algorithm. The topic will be defined and explained in Chapter 5. At last, the fourth thesis question is treated in Chapter 6, in which different scenarios have been used to assess the AI-based guidance and control methodology both in terms of performance and in terms of method definition and exploitation. This part is cross-sectional to the different blocks, as outlined in Fig. 1.1.

1.3 Bibliographic Disclaimer

The work presented in this thesis is the result of three years of Ph.D., during which different research activities were carried out. When the opportunity made it possible, the products of these activities have been published either in conference or journal papers. As such, some parts of this thesis have already been presented in previous articles, reported in the following list.

- **Andrea Brandonisio**, Lorenzo Capra and Michèle Lavagna. *Deep Reinforcement Learning Spacecraft Guidance with State Uncertainty for Autonomous Shape Reconstruction of Uncooperative Target.* Advances in Space Research. 2023. doi: `https://doi.org/10.1016/j.asr.2023.07.007`

- Lorenzo Capra, **Andrea Brandonisio** and Michèle Lavagna. *Network Architecture and Action Space Analysis for Deep Reinforcement Learning towards Spacecraft Autonomous Guidance.* Advances in Space Research. 2022. doi: `https://doi.org/10.1016/j.asr.2022.11.048`

- **Andrea Brandonisio**, Lorenzo Capra and Michèle Lavagna. *Spacecraft Adaptive Deep Reinforcement Learning Guidance with Input State Uncertainties in Relative Motion Scenario.* 2023 AIAA SciTech Forum, National Harbor, MD, USA, 23-27 January, 2023.

- Lorenzo Capra, **Andrea Brandonisio**, Michèle Lavagna. *Adaptive Vision-Based Autonomous Guidance with Deep Reinforcement Learning.* 11^{th} International Workshop on Satellite Constellations & Formation Flying, Milan, Italy, 7-10 June 2022.

- **Andrea Brandonisio**, Michèle Lavagna and Davide Guzzetti. *Reinforcement Learning for Uncooperative Space Objects Smart Imaging Path-Planning.* The Journal of Astronautical Sciences. 2021. doi: `https://doi.org/10.1007/s40295-021-00288-7`

- **Andrea Brandonisio** and Michèle Lavagna. *Sensitivity Analysis of Adaptive Guidance via Deep Reinforcement Learning for Uncooperative Space Object Imaging.* 2021 AAS/AIAA Astrodynamics Specialist Conference Big Sky, 08-12 August 2021.

In the following, instead, the list of supervised related works for students' Master's Degree Thesis:

- Matteo D'Ambrosio. *Deep Reinforcement Learning Aided Robotics for Uncooperative Space Asset Grasping and In-Orbit Servicing.* Prof. Michèle Lavagna, Dicember 2023, A.Y. 2022/2023.

- Davide Iafrate. *Assessment of Deep Reinforcement Learning for flexibility enhancement of planetary landing guidance and control*. Prof. Michèle Lavagna, May 2023, A.Y. 2022/2023.
- Lorenzo Capra. *Adaptive Vision-Based Autonomous Guidance with Deep Reinforcement Learning*. Prof. Michèle Lavagna, April 2022, A.Y. 2021/2022.
- Nicole Testa. *Reinforcement Learning for the Autonomous Planning of Debris Collision Avoidance Strategies in Densely Populated Environments*. Prof. Michèle Lavagna, July 2021, A.Y. 2020/2021.

CHAPTER 2

Background & State-of-the-Art

> Noi sapevamo che vecchia era la lunga guerra che stavamo combattendo, e che in un giorno l'avrebbe vinta chi sarebbe stato capace di combatterla in un modo nuovo.
>
> — Alessandro Baricco, *Omero, Iliade*

This chapter lays the foundations for the future main problem and scenario development. In particular, here, the background of this thesis is analysed in terms of State-of-the-Art (SOTA) of the different parts, which characterize the thesis' main problem: chaser-target relative dynamics motion (Section 2.1), Deep Reinforcement Learning for spacecraft guidance and control (Section 2.2), and image processing techniques for object's map reconstruction (Section 2.3). Since also the software selection is strongly important in machine learning applications, in Section 2.4, a brief overview of the coding software is presented.

2.1 Dynamical Model

In the following section, the main coordinate reference frames and physical formulations of the dynamical models used in the thesis are presented. Because the main problem will be defined in a relative motion scenario, in the following subsections, the models are differentiated between translational and rotational.

2.1.1 Coordinate Reference Frames

Before entering into the details of the mathematical model upon which the environment is built, it is important to define the several reference frames in which the dynamical models will be defined in the following sections. The objects involved in the scenario that will be analysed are mainly two: the controlled spacecraft, defined also as Chaser (CHS), and the space object, defined as Target (TRG). In this thesis, the chaser represents the inspecting spacecraft and the target is the unknown and uncooperative object. During the work, the following four reference frames will be used as defined in [27]:

- **ECI-reference frame.** This is the system which has the origin in the centre of the Earth and the plane as the Earth's equator (Earth-centerd Inertial (ECI)). The $\hat{I}$ is directed along the vernal equinox, and the $\hat{K}$ points towards the Earth's North Pole; the $\hat{J}$ completes the right-handed triad. This reference frame will be declared for both chaser and target as ECI_{CHS} and ECI_{TRG}.

- **LVLH-reference frame.** This is the Local Verical, Local Horizontal (LVLH) reference frame, which moves with the satellite and has the origin in the satellite's center of mass (CM). The typical definition of this reference frame fixes the $\hat{i}$ and $\hat{j}$ ad the two axes laying on the satellite's orbital plane, with the first pointing out from the satellite along the Earth's radius vector (Local Vertical) and the second towards the velocity vector (Local Horizontal). Instead, the third axis $\hat{k}$ is normal to the plane. Nevertheless, in this work, a slightly different, even if quite common, formulation will be used: $\hat{i}$ points from to satellite towards the center of the Earth, $\hat{j}$ points towards the velocity vector, and, therefore, the $\hat{k}$ is normal to the plane but pointing the opposite direction of the orbital angular-momentum. From now on, as LVLH frame is intended the one here described. As for the ECI reference frame, also this one will be declared for both chaser and target as LVLH_{CHS} and LVLH_{TRG}.

- **CHS-reference frame.** This reference frame is centred in the spacecraft CM and represents its Cartesian right-hand body coordinate system, in which the axes are aligned parallel to the principal axis of inertia. In the following analysis, for the sake of simplicity, the camera on the chaser will

be aligned with one of the principal body axes. Therefore CAM-frame will be considered equal to CHS-frame.

- **TRG-reference frame.** Alike the chaser CHS-frame, this frame represents the body coordinate system centred in the target CM; also here, it is assumed that the system coincides with the three principal axes of inertia.

The main part of the work is based on the assumption that the chaser is always target pointing and that the camera is always directed towards the target object CM, afterwards discussed in detail in Sec. 3.2. This simplification, when valid, lightens the computation (which is a very important aspect when dealing with very long simulations) and the mathematical approach needed to define the chaser-target relative dynamics. In conclusion, when the target pointing assumption holds, the CHS-frame is unnecessary.

2.1.2 Translational Dynamical Model

As already introduced before, most of the work is based on a relative-dynamic scenario, that can be divided into two uncoupled behaviours: the translational and rotational ones. Here a non-linear eccentric model of the translational relative motion is described starting from the Keplerian Two-body problem (2BP) under some particular assumptions, as explained in [28]. According to Newton's Second Law, considering no external or internal influences except the gravitational force, spherical bodies, no tidal forces, and mass of the attractor body much larger than the orbiting body mass, the equations of motion can be written as:

$$\ddot{\mathbf{r}} + \mu \frac{\mathbf{r}}{r^3} = 0 \tag{2.1}$$

where $\mathbf{r} = [r_x, r_y, r_z]^T$ is the vector of the orbiting body position defined in the inertial reference frame (ECI), μ is the main attractor's gravitational constant, and $r = \|\mathbf{r}\|$ is the position's module. This equation can be written both for the CHS and TRG in the ECI reference frame:

$$\ddot{\mathbf{r}}_{CHS} + \mu \frac{\mathbf{r}_{CHS}}{r_{CHS}^3} = 0 \tag{2.2}$$

$$\ddot{\mathbf{r}}_{TRG} + \mu \frac{\mathbf{r}_{TRG}}{r_{TRG}^3} = 0 \tag{2.3}$$

Once defined the objects' main motion in Eq. 2.2 and Eq. 2.3, it is useful to compute the relative position and velocity in the comoving reference frame (LVLH defined before), exploiting the following relation:

$$\hat{i} = \frac{\mathbf{r}}{r}, \;\; \hat{k} = \frac{\mathbf{h}}{h}, \;\; \hat{j} = \hat{k} \times \hat{i}$$

where $\mathbf{h}$ is the angular momentum. In this way, it is possible to declare the chaser-target relative position both in ECI:

$$\delta\mathbf{r} = \mathbf{r}_{TRG} - \mathbf{r}_{CHS} \tag{2.4}$$

and in LVLH reference frames:

$$\boldsymbol{\rho} = x\hat{i} + y\hat{j} + z\hat{k} \tag{2.5}$$

Then, subtracting Eq. 2.3 to Eq. 2.2, and substituting Eq. 2.4, the following relative acceleration can be retrieved in ECI reference frame:

$$\delta\ddot{\mathbf{r}} = -\frac{\mu(\mathbf{r}_{CHS} + \boldsymbol{\rho})}{\|\mathbf{r}_{CHS} + \boldsymbol{\rho}\|^3} + \mu\frac{\mathbf{r}_{CHS}}{r_{CHS}^3} \tag{2.6}$$

In general, the relative acceleration, $\ddot{\boldsymbol{\rho}} = \ddot{x}\hat{i} + \ddot{y}\hat{j} + \ddot{z}\hat{k}$, in the comoving frame can be expressed in function of the absolute relative acceleration as following:

$$\delta\ddot{\mathbf{r}} = \frac{d^2\boldsymbol{\rho}}{dt^2} + 2\boldsymbol{\omega} \times \frac{d\boldsymbol{\rho}}{dt} + \frac{d\boldsymbol{\omega}}{dt} \times \boldsymbol{\rho} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{\rho}) \tag{2.7}$$

where $\boldsymbol{\omega}$ is the angular velocity of the chaser LVLH frame with respect to the ECI, and the Coriolis acceleration $2\boldsymbol{\omega} \times \frac{d\boldsymbol{\rho}}{dt}$, the Euler $\frac{d\boldsymbol{\omega}}{dt} \times \boldsymbol{\rho}$ and the centrifugal $\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{\rho})$ accelerations are added to the relative acceleration in the moving frame. Afterwards, combining Eq. 2.6 and Eq. 2.7, under the assumption of angular velocity $\boldsymbol{\omega} = [0, 0, \dot{\theta}_{CHS}]^T$ normal to the orbital plane in LVLH frame, recalling Eq. 2.4, the following system of non-linear second order differential equation can be retrieved:

$$\begin{cases} \ddot{x} = 2\dot{\theta}_{CHS}\dot{y} + \ddot{\theta}_{CHS}y + \dot{\theta}_{CHS}^2 x - \dfrac{\mu(r_{CHS} + x)}{[(r_{CHS} + x)^2 + y^2 + z^2]^{3/2}} + \dfrac{\mu}{r_{CHS}^2} \\ \ddot{y} = -2\dot{\theta}_{CHS}\dot{x} - \ddot{\theta}_{CHS}x + \dot{\theta}_{CHS}^2 y - \dfrac{\mu y}{[(r_{CHS} + x)^2 + y^2 + z^2]^{3/2}} \\ \ddot{z} = -\dfrac{\mu z}{[(r_{CHS} + x)^2 + y^2 + z^2]^{3/2}} \end{cases} \tag{2.8}$$

where $\dot{\theta}_{CHS}$ and $\ddot{\theta}_{CHS}$ are respectively the angular velocity and acceleration of the leader. which values can be computed as:

$$\dot{\theta}_{CHS} = \sqrt{\frac{\mu}{a_{CHS}^3(1-e_{CHS}^2)^3}}(1+e_{CHS}\cos\theta_{CHS})^2 = \frac{h_{CHS}}{r_{CHS}^2} \tag{2.9}$$

$$\ddot{\theta}_{CHS} = -2\frac{\mathbf{v}_{CHS}\cdot\mathbf{r}_{CHS}}{r_{CHS}^2}\dot{\theta}_{CHS} \tag{2.10}$$

Even if this mathematical formulation relies on some assumptions because both r_{CHS} and ω are time-dependent, it needs additional differential equations to be solved; moreover, it can be complex and heavy to computationally solve the motion because of the fractional term that could explode or nullify due to the great difference in order of magnitude between the relative coordinates $[x, y, z]$ and the attractor distance r_{CHS}.

To overcome this computational problem, the linearization of the equations may be a viable option. This procedure is based on the fact that the relative distance between the chaser and the target, in any case, is much lower than the distance between the two and the main attractor, $\rho/r_{TRG} << 1$ and $\rho/r_{CHS} << 1$, where $\rho = \|\boldsymbol{\rho}\|$ and $r_{CHS} = \|\mathbf{r}_{CHS}\|$, $r_{TRG} = \|\mathbf{r}_{TRG}\|$. Such an assumption is reasonable considering the typical application in which relative dynamics is commonly used, e.g. close-proximity. Therefore starting from Eq. 2.6, computing the Taylor expansion, the relative acceleration in the inertial reference frame is:

$$\ddot{\boldsymbol{\rho}} \approx -\frac{\mu}{r_{CHS}^3}\left[\boldsymbol{\rho} - \frac{3}{r_{CHS}^2}(\mathbf{r}_{CHS}\cdot\boldsymbol{\rho})\mathbf{r}_{CHS}\right] \tag{2.11}$$

in which all the terms in $\frac{\rho}{r_{CHS}}$ higher then order one have been neglected. Therefore, as before, substituting Eq. 2.11 in Eq. 2.7, it is possible to define the relative motion in the LVLH reference frame as a system of linear second-order differential equations. These linearized equations can be written both in the LVLH frame centred in the chaser or the target. As in [29], a common use is to express the equations in the object LVLH reference frame. In general, the following linear system is obtained:

$$\begin{cases} \ddot{x} = \left(\dfrac{2\mu}{r^3} + \dfrac{h^2}{r^4}\right)x - \dfrac{2(\mathbf{v}\cdot\mathbf{r})h}{r^4}y + \dfrac{2h}{r^2}\dot{y} \\ \ddot{y} = \left(\dfrac{h^2}{r^4} - \dfrac{\mu}{r^3}\right)y + \dfrac{2(\mathbf{v}\cdot\mathbf{r})h}{r^4}x - \dfrac{2h}{r^2}\dot{x} \\ \ddot{z} = -\dfrac{\mu}{r^3}z \end{cases} \tag{2.12}$$

Where $\mathbf{r}, \mathbf{v}, r$ and h are respectively the position and velocity vector, position distance and angular momentum of the object in ECI reference frame centred in the main attractor. For a complete overview of the mathematical assumptions and steps, please refer to [28]. In general, this formulation works for eccentric reference orbits and is the most general one for unperturbed relative motion.

If the reference orbit is circular (or almost circular) the Linearized Eccentric Model defined in Eq. 2.12 can be considerably simplified. Indeed, when the eccentricity $e = 0$, the scalar product $\mathbf{v} \cdot \mathbf{r} = 0$, because the position and velocity vector are always perpendicular. Therefore knowing that $h = \sqrt{\mu R}$, Eq. 2.12 becomes:

$$\begin{cases} \delta\ddot{x} - 3\frac{\mu}{r^3}\delta x - 2\sqrt{\frac{\mu}{r^3}}\delta\dot{y} = 0 \\ \delta\ddot{y} + 2\sqrt{\frac{\mu}{r^3}}\delta\dot{x} = 0 \\ \delta\ddot{z} + \frac{\mu}{r^3}\delta z = 0 \end{cases} \tag{2.13}$$

Furthermore, knowing that for a circular orbit the mean motion (that it is equal to the angular velocity) can be written as follows:

$$n = \frac{v}{r} = \frac{\sqrt{\mu/r}}{r} = \sqrt{\frac{\mu}{r^3}} \tag{2.14}$$

the equations in the system in Eq. 2.13 becomes the well-known Clohessy-Wiltshire Model (CW) equations [30]:

$$\begin{cases} \delta\ddot{x} - 3n^2\delta x - 2n\delta\dot{y} = 0 \\ \delta\ddot{y} + 2n\delta\dot{x} = 0 \\ \delta\ddot{z} + n^2\delta z = 0 \end{cases} \tag{2.15}$$

In this case, all the coefficients are constant, thus an analytical solution exists and can be written as:

$$\begin{cases} \delta x = 4\delta x_0 + \frac{2}{n}\delta\dot{y}_0 + \frac{\delta\dot{x}_0}{n}\sin nt - \left(3\delta x_0 + \frac{2}{n}\delta\dot{y}_0\right)\cos nt \\ \delta y = \delta y_0 - \frac{2}{n}\delta\dot{x}_0 - 3(2n\delta x_0 + \delta\dot{y}_0)t + 2\left(3\delta x_0 + \frac{2}{n}\delta\dot{y}_0\right)\sin nt + \frac{2}{n}\delta\dot{x}_0\cos nt \\ \delta z = \frac{1}{n}\delta\dot{z}_0\sin nt + \delta z_0\cos nt \end{cases} \tag{2.16}$$

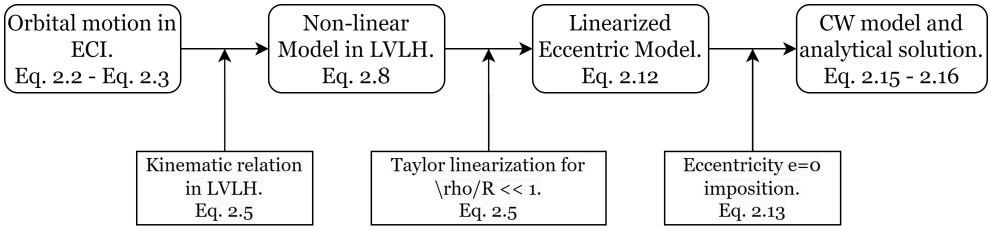


Figure 2.1: Conceptual process from equations of motion in ECI, to CW model in LVLH.

All three components oscillate with a frequency equal to the frequency of the revolution of the CW frame, while the only secular term is in δy, which grows linearly during the time, meaning that the spacecraft will drift away from the target vehicle if the initial condition is $2n\delta x_0 + \delta \dot{y}_0 \neq 0$. The overall process, from the equation of motion in the ECI reference frame, to the analytical solution of the CW system in LVLH is schematized in Fig. 2.1.

It is important to underline why the choice of the Linearized Eccentric model suits better than the CW model for the scenario analysed in the following chapters. Indeed, as shown in Fig. 2.2 and Fig. 2.3, from a very simple comparative analysis between the two models, it is easy to understand how much the orbit ground-truth can change when the reference orbit present an eccentricity not null, even if quite small. The trajectories shown are integrated starting from the same initial conditions; the reference orbit's eccentricity is equal to 0.1 for the Linearized Eccentric model. As observable, the CW trend is stable and constant in time without degenerating as for the linearized eccentric one: in particular, the x and y components are differentiated by a continuous change both in phase and amplitude, while the z is only affected by an increasing phase-shift. This behaviour can lead to quite different trajectories, that may slightly or greatly influence the results obtained by the learning agent, as treated in the following chapters.

Moreover, maintaining a certain level of generalization in the target object reference orbit around the main attractor is preferable for a reinforcement learning application, which typically has its strength right in the generalization capability. The limitation on the eccentricity forced by the CW model is seen as too restrictive, and therefore discarded; thus, for this reason, the Linearized Eccentric model is adopted as the mathematical model for the translational relative dynamic.

In the AI-based GNC pipeline, which will be discussed in Sec. 5, the translational dynamics of the chaser and target objects will not be based on the integration of a relative-dynamics system as an abode, but will directly come from the integration of the 2BP, defined in Eq. 2.1 with the addition of the

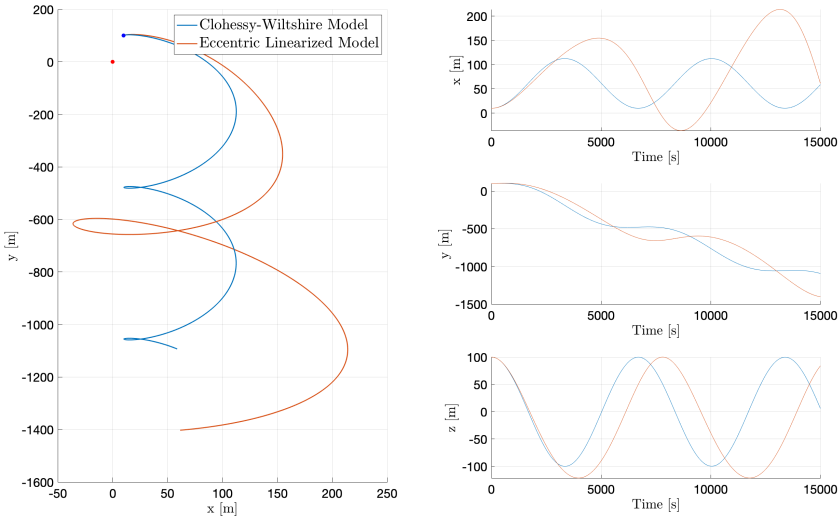


Figure 2.2: Comparison between CW (e = 0) and Linearized Eccentric models (e = 0.1) with initial conditions: $\mathbf{r}$ = [10, 100, 10] m, $\mathbf{v}$ = [0 0.01 0.001] m/s [26]

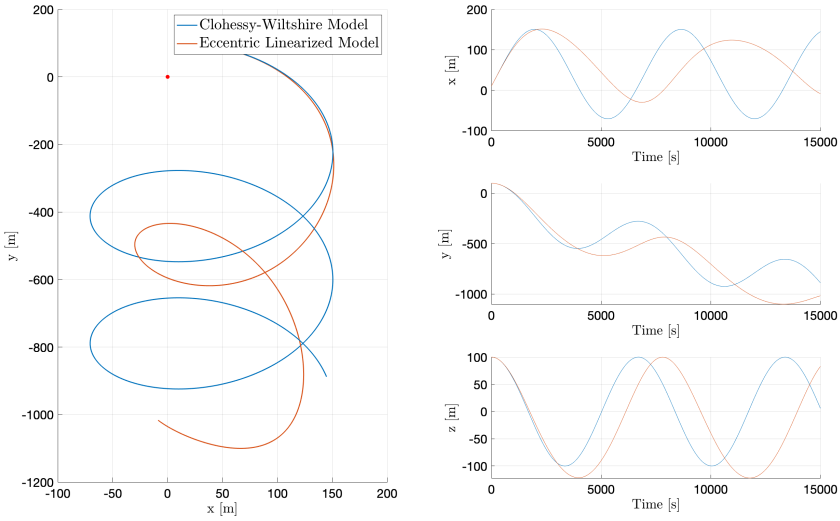


Figure 2.3: Comparison between CW (e = 0) and Linearized Eccentric models (e = 0.1) with initial conditions: $\mathbf{r}$ = [10, 100, 100]m, $\mathbf{v}$ = [0.1 0 0.001] m/s [26]

perturbing gravitational acceleration $\mathbf{b}$ due to J_2, defined as follows in the ECI reference frame:

$$\mathbf{p} = \frac{3}{2}\frac{J_2 \mu R^2}{r^4}\left[\frac{x}{r}\left(5\frac{z^2}{r^2}-1\right)\hat{\mathbf{I}} + \frac{y}{r}\left(5\frac{z^2}{r^2}-1\right)\hat{\mathbf{J}} + \frac{z}{r}\left(5\frac{z^2}{r^2}-3\right)\hat{\mathbf{K}}\right] \quad (2.17)$$

Where R is the Earth's mean radius equal to 6378 km. For a complete mathematical formulation of the dynamics, please refer to [29].

2.1.3 Rotational Dynamical Model

The relative-dynamic scenario, which will be treated in the following chapters, demands also the formulation of a rotational dynamic. It is quite commonly used to express the rotational dynamics of a body through the Euler's equations defined in body-reference frame [31], CHS-frame for the chaser, and TRG-frame for the target object, as in Eq. 2.18.

$$\begin{cases} I_x\dot{\omega}_x + (I_z - I_y)\omega_y\omega_z = 0 \\ I_y\dot{\omega}_y + (I_y - I_z)\omega_z\omega_x = 0 \\ I_z\dot{\omega}_z + (I_x - I_x)\omega_x\omega_y = 0 \end{cases} \quad (2.18)$$

Where I_x, I_y, I_z are the moments of inertia of the body along the three principal axes, and $\omega_x, \omega_y.\omega_z$ the three components of the angular velocity of the body around the three axes.

For the future formulation of the scenario, it could be very useful to potentially define both the relative rotational dynamics in the same reference frame of the relative translational dynamics, which is the LVLH frame centred in the object **c.m.!**. At this point, it is necessary to create a system of angles also in the LVLH frame. This is possible under the assumption of small angles. Therefore, considering a rotation around the z-axis and a set of Euler angles with three different indexes, to avoid singularity in the nominal condition, being their order not relevant, the rotation matrix representing the relative attitude of the satellite in the LVLH frame is:

$$A = \begin{bmatrix} 1 & \alpha_z & -\alpha_y \\ -\alpha_z & 1 & \alpha_x \\ \alpha_y & -\alpha_x & 1 \end{bmatrix} \quad (2.19)$$

Afterwards, using the transpose theorem, it is possible to write the angular velocities and accelerations in the body frame exploiting the rotations in LVLH frame:

$$\begin{Bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{Bmatrix} = \begin{bmatrix} \dot{\alpha}_x \\ \dot{\alpha}_y \\ \dot{\alpha}_z \end{bmatrix} + \begin{bmatrix} 1 & \alpha_z & -\alpha_y \\ -\alpha_z & 1 & \alpha_x \\ \alpha_y & -\alpha_x & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ n \end{bmatrix} \tag{2.20}$$

$$\begin{cases} \dot{\omega}_x = \ddot{\alpha}_x - \dot{\alpha}_y n \\ \dot{\omega}_y = \ddot{\alpha}_y + \dot{\alpha}_x n \\ \dot{\omega}_z = \ddot{\alpha}_z \end{cases} \tag{2.21}$$

Where n is the nominal angular velocity of the body along its orbit. Then, substituting $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ in the Euler's rotation equations for free-torque motion Eq. 2.18, the differential system of equations for the rotational dynamic of the body in LVLH can be retrieved as follows:

$$\begin{cases} I_x(\ddot{\alpha}_x - \dot{\alpha}_y n) + (I_z - I_y)(\dot{\alpha}_z + n)(\dot{\alpha}_y + \alpha_x n) = 0 \\ I_y(\ddot{\alpha}_y + \dot{\alpha}_x n) + (I_x - I_z)(\dot{\alpha}_z + n)(\dot{\alpha}_x - \alpha_y n) = 0 \\ I_y\ddot{\alpha}_z + (I_y - I_x)(\dot{\alpha}_x - \alpha_y n)(\dot{\alpha}_y + \alpha_x n) = 0 \end{cases} \tag{2.22}$$

Under the small angles assumption, the products between infinitesimal terms can be neglected, thus leading to the final system in Eq. 2.23.

$$\begin{cases} I_x\ddot{\alpha}_x + n(I_z - I_y - I_x)\dot{\alpha}_y + n^2(I_z - I_y)\alpha_x = 0 \\ I_y\ddot{\alpha}_y + n(I_x + I_y - I_z)\dot{\alpha}_x + n^2(I_z - I_x)\alpha_x = 0 \\ I_z\ddot{\alpha}_z = 0 \end{cases} \tag{2.23}$$

These equations of motion represent the attitude of the body in LVLH frame. In Fig. 2.4 the Euler's angles $\boldsymbol{\theta}$ and the LVLH angles $\boldsymbol{\alpha}$ are shown, where α_x, α_y and α_z are the angles around the three directions: opposite-yaw, roll and pitch. The order in which the LVLH angles have been taken is not relevant to have the equations effective and representative of the model. Therefore the reader should not be surprised by the directions shown in Fig. 2.4, which do not coincide with the common right-hand rule.

Differently, no attitude dynamics have been considered for the spacecraft. In this way, the relative rotational model is much easier to formulate and solve. This simplification was possible because it seemed reasonable to assume

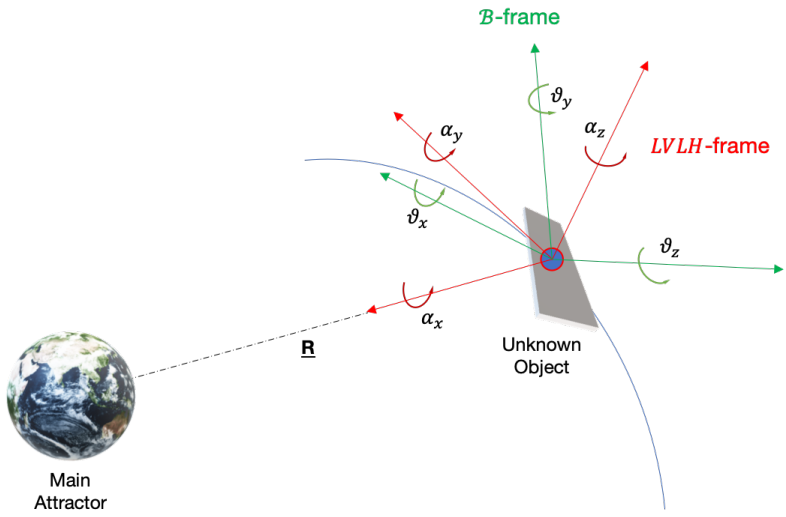


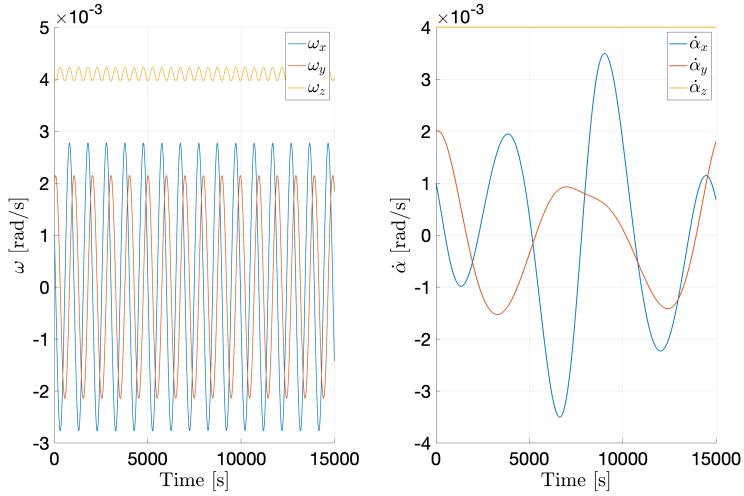
Figure 2.4: Euler's angles and LVLH angles

the spacecraft continuously pointing at the center of the object. Therefore any attitude manoeuvrer and control performed by the spacecraft has been neglected. The reasons behind this choice are treated more in detail in Sec. 3.2.

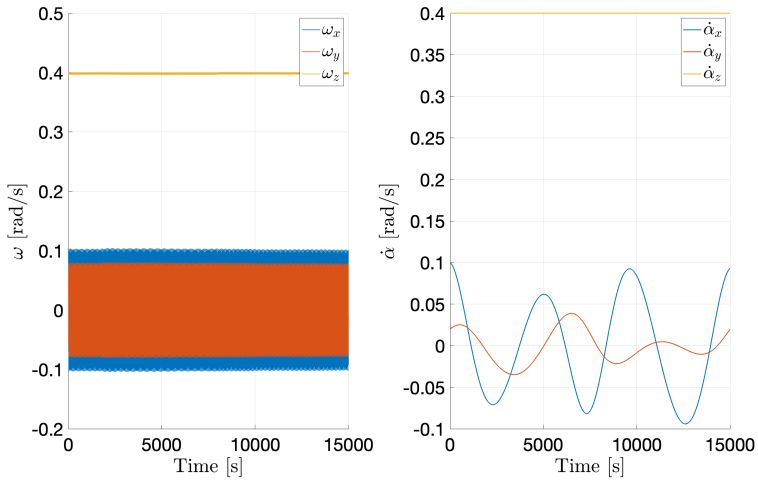
In Fig. 2.5 the trend of both Euler angular velocities and LVLH angular velocities is shown. Two cases are presented in order to underline the differences between a small angle case and a high angle case - in which the approximation can not exist anymore. This plot can also be used as a sort of justification for having used the LVLH attitude angles and not the usual Euler angles. Indeed, as can be seen, the temporal variation of the last ones is very fast compared to that in the LVLH frame. Therefore, considering the state composed by these angles as possible input for an artificial intelligence agent, the fast-variation characteristic theoretically can slow down the learning process, which can be favoured by a slow-varying parameter. However, it is worth pointing out that from a computational point of view, using the non-linearized model for the derivation of the LVLH angular velocities differential equations would not change practically anything. Nevertheless, this choice has been made always pointing toward the direction of simplifying most of the learning environment that must be taught to the agent. Indeed, a forward step in the work could be the one of considering any kind of rotation.

2.2 Deep Reinforcement Learning

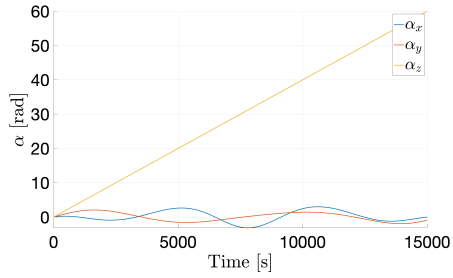
Before entering into the detail of the deep reinforcement learning techniques used for the development of the analysis, it is worth spending some words on the overall *machine learning* frameworks: this will help also in understanding the



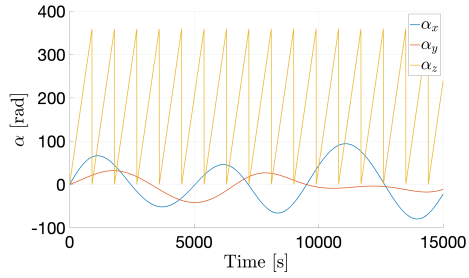
(a) $\boldsymbol{\omega} = [0.001, 0.002, 0.004]$rad/s $\dot{\boldsymbol{\alpha}} = [0.001\ 0.002\ 0.001]$rad/s



(b) $\boldsymbol{\omega} = [0.1, 0.02, 0.4]$rad/s $\dot{\boldsymbol{\alpha}} = [0.1\ 0.02\ 0.1]$rad/s



(c) small angles



(d) big angles

Figure 2.5: Comparison of Euler's velocities and LVLH models for small and big angles [26].

decision of exploiting reinforcement learning in this work. ML approaches can be classified into three categories, which depend on the nature of the *feedback* available by the learning system. They are supervised learning, unsupervised learning and reinforcement learning.

Nowadays, the research is mainly focused on supervised learning, which makes use of a training set of examples provided by an external supervisor; all of them represent a particular description of a situation in a specific environment, intended as the correct action that the system has to take when it finds itself in a particular state. The final objective of training an agent with supervised learning is the achievement of the agent awareness, aiming to have the agent capable of correctly acting also in situations that have been not experienced in the training set. This methodology is very powerful for solving problems such as image recognition, on the contrary, it is inadequate for learning from interaction. Indeed, in highly interactive problems it is rarely possible to obtain a training set covering the desired behaviour for all the situations in which the agent could act. For this kind of problem, the best way for the agent to learn is by doing it from its own experience. This path is followed by the second branch of ML called unsupervised learning, which learns to find hidden structures in collections of *unlabeled* data. This means that, if for supervised learning the agent is fed by example inputs and desired outputs, in unsupervised learning the agent is left on its own to find the structure in given inputs because no a priori knowledge is presented as outputs.

Even if these two methodologies seem to exhaustively classify the machine learning paradigms, the RL branch is the last component of this picture. Differently from unsupervised learning, RL tries to maximize a reward instead of finding hidden structures from a set of given data. The two important characteristics of this method, that distinguish it from all the others, are the trial-and-error search and the reward. In fact, the *learning* agent starts without any knowledge about the optimal action to take for a particular problem, and therefore it is forced to discover which are the ones that yield the greatest reward number. Each action taken may affect not only the immediate reward but also the future development of the problem and, then, the overall sequence of rewards. These concepts outlined here represent the fundament of RL. and are mathematically encapsulated in the Markov Decision Process (MDP) problems [24], that can be fully or partially observable (as explained later). The final objective of MDP is finding a policy capable of selecting the right action when the agent is in a particular state. Policies may also be found with evolutionary methods, but they are capable of achieving a feasible solution only when the search space is small, otherwise, the optimization becomes computationally intractable. A Markov Decision Process has three main aspects: sensation, action and goal. The agent is called to sense the state in its environment and

to take an action that affects the state itself, in order to achieve the single or multiple goals, it is meant to accomplish.

RL differs from all others *learning* methods since its formulation requires a trade-off between *exploration* and *exploitation*, a choice that represents one of the most challenging features of the methodology. This dichotomy is straightforward to understand: indeed, it is easy to agree on the fact that, to obtain a good reward, an agent should prefer those actions already tried in the past and that have been found to be effective for the final reward value. But, on the contrary, to find such kinds of actions, the same agent should explore actions that have been never selected before. This means that, at each decisional step the agent has the dilemma of exploiting already experienced situations that bring sure results, or exploring new situations to find better actions for the future. This exploitation-exploration competition, in general, generates an RL agent which wants to try a variety of actions, progressively favouring the ones that appear to work better.

Once understood these key concepts, it may be easy to comprehend the main elements of a generic reinforcement learning system that, beyond the agent and the environment, are the policy, the reward function, the value function and, if present, the model of the environment itself.

- *Policy*. It defines how the agent behaves at a given time. In practical terms, this learned behaviour can be considered as a map from the states to the actions that can be taken in an environment. It can be a simple function or a very complex element that requires extensive computations, but in general, it is a stochastic element that specifies the probability of taking an action when the agent is in a particular state.

- *Reward*. The reward is the function defining the aim of the problem that the agent must optimize in the environment. As already underlined, if the agent's objective is the maximization of the total reward, the reward function is a mathematical representation of what is good or bad for the agent in an immediate sense; it is the primary reason based on which the policy changes over time. In general, the reward is formulated in function of the state of the environment and of the actions.

- *Value function*. In contrast to the reward function, the value function determines what is good in the long run. It is defined as the total amount of rewards that an agent can expect to accumulate in the future, starting from the state in which it is computed. So, if on the one hand rewards are the immediate desirability of the state, on the other hand, the *values* are the long-term desirability of states after considering all the ones that are likely to follow, coupled with the rewards achievable in those subsequent

states. This means that a state may yield a low reward but have a high *value* because it is followed by other states that bring higher rewards.

- *Model*. The model is the element that simulates the real behaviour of the environment. It is used as a planning component by which an action can be decided, based on the knowledge of the possible future situations, before the agent actually experiences them. Reinforcement learning methods that use prebuilt environment models are called *model-based* methods, otherwise, they are *model-free* methods. This distinction defines two separate branches of RL algorithms: the model-based methods are characterized by agents that exploit predictive planning thanks to the knowledge of the system dynamics, they need minimal use of *exploration*, at the expense of having a higher computational cost while executing. The model-free methods, instead, do not have any hint of the dynamics, and therefore require extensive use of *exploration*, even if the computation cost is lighter. For this reason, in the context of spacecraft guidance and control, the model-based approach seems to suit better and be beneficial [6]. Nevertheless, as treated in Section 2.2.1, both methodologies are used in RL applications for spacecraft guidance and control.

In the next sections, a brief introduction about the state-of-the-art application of DRL for spacecraft guidance and control will be presented. Afterwards, DRL will be treated more in detail, starting from the formulation of the mathematical processes for which DRL is proposed as a solution. Then, because DRL needs a powerful function approximator to define the policy, ANN will be introduced and therefore discussed especially focusing the attention on the different models that can be useful for the main scenario that is analysed in this work. In the end, the most interesting DRL algorithms will be outlined, underlining the main properties that drive their choice.

2.2.1 Guidance and Control with DRL

As anticipated in Chapter 1, an active research field is to use reinforcement learning and meta-reinforcement learning (meta-RL) to create an adaptive guidance and control system [3, 25]. Deep reinforcement learning has been used to generate autonomous guidance and control during proximity operations and landing trajectories by Brandonisio [26] and Ciabatti *et al.* [32]. Reinforcement learning has been used to generate autonomous trajectory planning for different scopes. Pesce *et al.* [13], Piccinin *et al.* [15], and Chan *et al.* [14] analysed the autonomous mapping of asteroids using DQN, Neural Fitted Q as value-based methods. Federici *et al.* [21] proposed an actor-critic PPO framework for real-time optimal spacecraft guidance during terminal rendezvous manoeuvrers, in the presence of both operational constraints and stochastic effects, such

as an inaccurate knowledge of the initial spacecraft state and the presence of random in-flight disturbances.

Brandonisio *et al.* [33] proposed a guidance and control law to perform the inspection of an uncooperative spacecraft, exploiting Deep Q-Network (DQN) and A2C methods, with the employment of MLPs as function approximators. A discrete action space is maintained, whereas the state space is continuous. Time Learning (TL) is also applied to facilitate training on more complex tests. Many researchers claim a superior performance of policy-based methods. Among those, PPO and derivatives are one of the most adopted schemes [34, 17, 18]. The work by Hovell *et al.* [5] proposes a guidance strategy for spacecraft proximity tracking operations leveraging deep reinforcement learning, in which the distributed distributional deep deterministic policy gradient (D4PG) [35] algorithm is used. Such an algorithm operates in continuous state and action spaces and has a deterministic output.

The robustness to uncertain spacecraft models and environments is a crucial topic in the development of autonomous systems. This aspect is very challenging when it comes to reinforcement learning methods. Neural networks learn well within the training distributions, but they generally fail when performing extrapolation outside the training distribution [36]. This may pose a risk of instability of the guidance law when the spacecraft experiences states that are outside the training distribution envelope. Several researchers [17, 37, 19] claim that sampling inefficiency is another weakness of traditional reinforcement learning: a large amount of experience is needed to learn even simple tasks.

Recent advancements in meta-RL have aimed at addressing these weaknesses of the traditional framework. Reinforcement meta-learning trains the policy agent on a distribution of environments or MDPs. This forces the agent to experience and learn multiple, and different, situations. Consequently, the system tends to converge faster to quasi-optimal solutions. Recent works claim superior performance of the meta-RL with respect to classical RL when uncertain environments and actuator failures are considered [17]. Gaudet *et al.* [19] developed a guidance law based on meta-RL which is able to perform six degrees-of-freedom Mars landing. Moreover, in [18], meta-RL is used to create a guidance law for hovering on irregularly shaped asteroids using light detection and ranging (LIDAR) sensor data. In certain works, the different environments are called tasks: the tasks can be thought of as the ensemble of potential situations, nominal and non-nominal, one can expect the agent to experience. For instance, in the application of meta-RL for planetary and asteroid landing, the tasks range from landing with engine failures to large mass variation or highly corrupted navigation and unknown dynamics. Li *et al.* [38] employed a meta-RL framework for relative trajectory planning between spacecraft. The training trajectories are divided into sub-training samples and fake testing

samples. The meta-reinforced agent is trained by alternating training and testing phases. To this end, the gradient information of the meta-learner is obtained through a combination of the results on the training subsets and the performance of the fake testing samples. The authors claim that this approach forces the agent to explicitly take account of the potential testing performance into consideration. In this way, the overfitting phenomenon, potentially arising using a few training trajectories, is reduced.

In most of the works, as for traditional reinforcement learning applications, the meta-RL policy is optimized using PPO, both the policy and value function implementing recurrent layers in their networks. Using RNNs results in creating agents that can adapt more easily to uncertain environments, yielding a much more robust guidance policy compared with classical reinforcement learning. Indeed, If considering a particular scenario such as a planetary landing, it is easier to understand how recurrent layers result in an adaptive agent. During the training to generate an autonomous landing agent, the next observation depends not only on the state and action but also on the GT agent mass and any external forces acting on the agent at each step. Consequently, during training, the recurrent layers force their hidden states to evolve differently depending on the observations acquired from the environment during the trajectory, which is governed by the actions output by the policy. Specifically, the trained policy's hidden state captures unobserved information such as external forces or agent mass that are useful in minimizing the cost function. Obviously, this holds for any scenario one could be interested in, even not related to space operations.

2.2.2 MDP and POMDP

As briefly introduced above, a MDP defines a decision-making problem in which an agent lives in a stochastic and sequential environment. Its formulation was first introduced in the late 50s by Bellman [39], and refined in the following years [40]. The essence of the process is that the agent inhabits an environment that changes according to the actions taken, and the state of this environment affects the reward signal as well as the probability of transitioning to a certain new state. As said, the final aim of the process is the maximization of the reward over time, also called *return*. MDP problems are characterized by the absence of any memory, known as *Markov property*, which means that all the rewards and transition probability depends exclusively on the current and next state. This property is fundamental for the mathematical formulation of the process but, most of the time is quite far from real application, where historical information drives the succession of states. Moreover, MDP imply the perfect knowledge of all the states of the environment. When this assumption is impossible to meet, a MDP problem turns into a POMDP problem. Substantially, POMDP is a MDP with state uncertainty, in which only a *belief* state is accessible to the agent using observations. For example, this formulation is valid whenever the

agent senses the environment via on-board sensors, which inherently introduce errors in the measurements.

A POMDP is characterized by more variables than MDP, in particular by a tuple of 6 elements (**S**, **A**, **R**, **T**, **Ω**, **O**), as introduced in [41]:

- **S** is the space of all possible states s in the environment;
- **A** is the space of all possible actions a that can be taken in all the states of the environment;
- **R** is the reward function, guiding the action selection to maximize it;
- **T** $(s_{k+1}|\, s_k, a_k)$ is the transition function governing the probability of moving from one state to the next, given the current state and action at timestep k;
- **Ω** is the space of possible observations;
- **O** $(o_{k+1}|\, a_k, s_{k+1})$ is the probability of making a particular observation, taking an action that leads to a particular new state.

There are several approaches to solve POMDP such as operations research methods [42], point-based [43], or machine learning algorithms. Nevertheless, this type of problem is often too complex to be solved and may become computationally intractable if not reduced to a simpler MDP. This approximation can be done by including the history h, which plays the role of an archive of past actions and observations. The new formulation, known as *belief-space* MDP, is described by a tuple of 4 elements (**B**, **A**, **R**, **T**):

- **B** is the belief space, where the *belief* is defined as $b = p(s|h)$, that is the probability of being in a certain state s after the history h.

Solving this kind of problem means computing a *policy* π, that defines the mapping function from states s to actions a that the agent is employing at each step k. This decision-making feature is "optimal" when the agent concurrently maximizes the reward function, which mathematically expresses the problem objectives. Thus, maximizing the reward signal received is equivalent to reaching the goal set by the designer, depending on the problem at hand. Two classes of problems can be distinguished by finite or infinite horizon problems, whose optimal policies are defined as in Eq. 2.24 and Eq. 2.25:

$$\pi_* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} R(b_{k+1}|b_k, a_k) \right] \tag{2.24}$$

$$\pi_* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(b_{k+1}|b_k, a_k) \right] \tag{2.25}$$

Where $\gamma \in [0, 1]$ is the so-called discount factor, introduced as a mechanism to control how short-sided is the agent, exponentially decaying the effect of rewards far away in time. R represents indeed the reward signal, depending on the action a and belief b at step k.

Considering the proposed work, a direct link between the elements describing a POMDP and the autonomous guidance problem characters can be highlighted:

- the agent is the chaser spacecraft, interacting with the surrounding environment, governed by the dynamics and the environment itself;
- the belief space b is computed from the sensors measurements, derived from the image processing and pose estimation steps;
- the action space a is the result of the available actuators' activity, such as the force exerted by switching on/off some spacecraft thrusters;
- the reward function strictly depends on the objectives, i.e. the shape reconstruction of the uncooperative and unknown target.

These scenario characteristics will be deeply treated and explained in Section 3.1.

2.2.3 Artificial Neural Network (ANN)

In Section 2.2, it has been explained how the optimization of a *policy* represents the core of every DRL application. So, if on the one hand DRL represents the mathematical approach with which POMDP problems are solved, on the other hand, it needs a mathematical tool capable of defining the policy. For this reason Neural Networks (NN) are a powerful tool for function approximation and, as such, they become very attractive in the DRL context to simulate the agent policy. ANN are a numerical tool derived from the structure of biological neural networks, that, after a training process, are capable of generating a specific output starting from a specific input. In the conventional programming approach, the programmer tells the computer what it has to do, breaking big problems up into many small tasks that the computer can easily perform. On the contrary, in ANN-based computer there is no knowledge on how to solve the problem but, instead, the computer learns it from observational data and experience. ANN are able to perform a large variety of tasks, such as function approximation, pattern association or recognition, control action and so on; they can be built with different architectures and trained by different algorithms [44]. In particular, regarding the training algorithms, three main categories exist: supervised, unsupervised and reinforcement learning.

As already underlined, in this work the attention is on the last kind of training. ANN have the ability to learn and model non-linear, complex relationships, generalizing the results, this means that they can infer input-output mappings

on unseen data. These key advantages make them a solid and robust candidate when in need of approximating a certain behaviour. In the following subsections, the two main models developed in this work will be characterized. Afterwards, the common training process used for ANN, known as *back-propagation* will be briefly explained.

The most basic, one-dimensional (1D) type of neural network is the well-known MLP, also called *feedforward, fully-connected* ANN, which means that the information flows from layer to layer always in the same direction, through a network that puts in connection every neuron of a layer to all the others of the next one. Also, more complex and popular MLP models exist, for instance, radial basis function neural networks (RBFNNs).

The other two macro types of ANNs are the CNNs, used for 2D input data, i.e., images, and RNNs, which instead are used for time series, or more in general, inputs that require state memory. All these models will be used or introduced in the following chapters to develop different parts of the AI-based GNC pipeline, in particular:

- **MLP** networks and **RNN**s will be exploited to develop the spacecraft's guidance and control tool, the main objective of the work presented in this thesis. Both the models will be treated in the next subsections and their architecture will be later discussed in Section 3.3.
- The **CNN**s, instead, is at the base of the Image Processing (IP) tool for space object pose estimation. The model will be briefly explained in Section 5.1.

2.2.3.1 MLP

The MLP are the most common and simple type of ANN. Their structure is made of several units called *neurons* or *perceptrons*, which are divided into different *layers*, that are connected through linear or non-linear relations [24, 45].

The first layer is the *input layer*, and the last layer, instead, is the *output layer*; all the layers in between are called *hidden layers*. The functioning of each single layer's *neuron* is straightforward: it takes several inputs $x_1, x_2, ..., x_N$ as the dimension of the previous layer and performs a weighted sum. Indeed, considering the i-th perceptron of the k-th layer of the network the output is computed as:

$$v_i^k = \sum_{j=0}^{N_{k-1}} w_{ij}^k x_j^k + b_i^k \tag{2.26}$$

Where w_{ij}^k is the weight from neuron j of the layer $k-1$ to neuron i of layer k and b_i^k is the bias of neuron i in layer k. The output computed in this way passes through a function f_i^k, called *activation function*, that generates the effective output of the perceptron.

$$y_i^k = f_i^k(v_i^k) \tag{2.27}$$

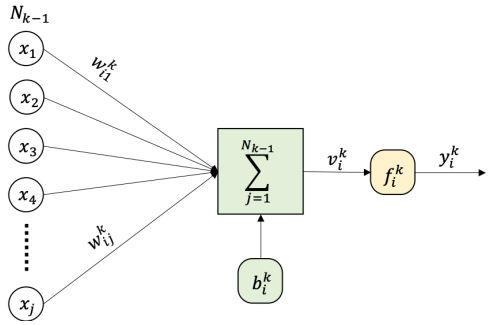


Figure 2.6: MLP neuron's output schematization.

Typical activation functions are the logic function, the sigmoid function, the linear function, the hyperbolic tangent function, and the rectified linear function (ReLU). As intuitive, the network can be structured as a sequence of layers, composed of a different number of neurons. Considering this process for the entire k layer, the input-output relation can be expressed as:

$$\mathbf{y}^k = f^k(\mathbf{W}^k\mathbf{x}^k + \mathbf{b}^k) \tag{2.28}$$

With $\mathbf{y}^k$ is the k-th layer output and has its length equal to the number of neurons in the layer N_k. $\mathbf{W}^k$ is the weight matrix, with dimensions $[N_k \times N_{k-1}]$.

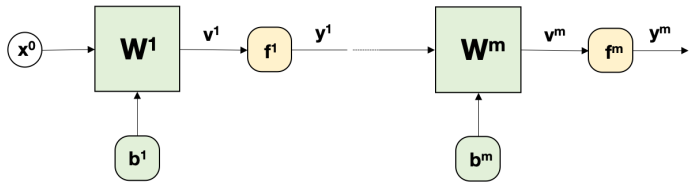


Figure 2.7: MLP network schematization.

Therefore, in general, the structure of an ANN is defined by different parameters, called *hyperparameters*, i.e. the number of hidden layers, the number of neurons per layer or the type of activation function. Their choice depends on the designer, even if some optimization procedures are also possible.

In Fig. 2.6 and Fig. 2.7, the schematization of how a single neuron and a multilayer network work is depicted.

2.2.3.2 RNN

Differently from feedforward, fully connected neural networks, that usually are used to solve classification and regression tasks involving fixed-size input vectors, and CNNs, which are suitable for complex image-related tasks, RNNs introduce loops. Both previous approaches are insufficient to tackle problems involving time series and sequence data, which require the ability to model temporal dynamics and long-term interactions. Instead, RNN's computations derived from earlier inputs are fed back into the network and then fed forward to be processed into outputs. Thus, they could take advantage of time correlation in the data and be more stable.

Commonly in ML, the tasks involving sequences are categorized by the length of the input and output sequences:

- *Many-to-one.* It maps a sequence of inputs to a single output.
- *One-to-many.* From a single input, it generates an output sequence.
- *Many-to-many.* Here, an input sequence is mapped to an output sequence of the same size. If an *encorder-decoder* structure is used, then, input and output sequences do not need to have the same size.
- *One-to-one.* If both the input and output sequences have length one, inputs can be considered independently without any interactions (such as MLP).

In order to process sequence data efficiently, RNNs tackle this problem by passing the activation from one timestep to the next. Taking into consideration a task in which an input sequence $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t$ is given and the network must predict the output sequence $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_t$, at time step t, the input vector $\mathbf{x}_t$ is passed to a hidden layer and the activations of the layer's units and the predicted output value $\mathbf{y}_t$ are calculated. Differently from MLP, the network layer now also receives the *hidden state* vector $\mathbf{h}_{t-1}$ containing the activations of the previous timestep as an additional input. The updated hidden state $\mathbf{h}_t$ and the output $\mathbf{y}_t$ are then obtained as:

$$\mathbf{h}_t = \boldsymbol{\sigma}_h(\mathbf{W}^x\mathbf{x}_t + \mathbf{U}^h\mathbf{h}_{t-1} + \mathbf{b}^h) \tag{2.29}$$

$$\mathbf{y}_t = \boldsymbol{\sigma}_y(\mathbf{W}^h\mathbf{h}_t + \mathbf{b}^y) \tag{2.30}$$

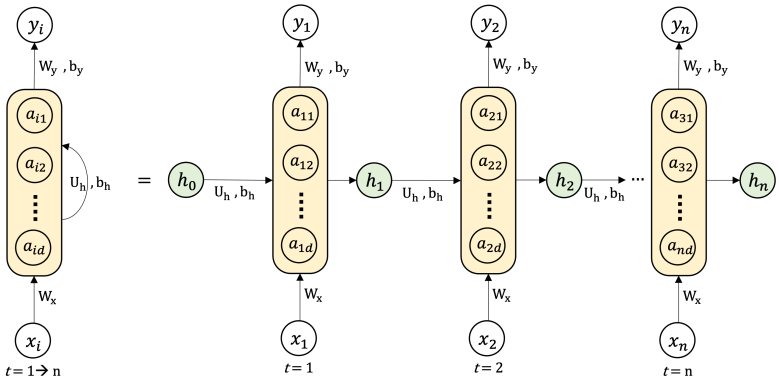


Figure 2.8: Schematic for a one-unit rRNN: compressed diagram on the left, *unfold* version on the right.

Where $\mathbf{W}^x, \mathbf{W}^h, \mathbf{U}^h$ are parameters matrices, $\mathbf{b}^h, \mathbf{b}^y$ are bias values vectors, and $\boldsymbol{\sigma}_h, \boldsymbol{\sigma}_y$ are the activation functions. The hidden state can be initialised ad $\mathbf{h}_0 = 0$. RNNs are often shown in the form of a compressed diagram in Fig. 2.8, where the feedback of the hidden state vector is depicted as a recurrent connection. The involved forward computations can also be represented for each timestep separately, or *unfolded* in time, making explicit the input, hidden state, and output at each step.

Analogously to feedforward NN, back-propagation through time is used to compute the derivatives of the total error with respect to all the states $\mathbf{h}$ and all the parameters in the unfolded RNN; *gradient descent* or other methods can, instead, be used for optimization [46].

RNN must be employed very carefully because training them has proved to be difficult due to the problem of *vanishing* or *exploding* gradients. Since the back-propagated gradients either grow or shrink at each time step, over many time steps, they can become very large (explode) or tend toward zero (vanish). Among the different types of RNN, in order to solve the issues of vanishing gradients, one of the first and widely used architectures is the Long-Short Term Memory (LSTM) network. It makes use of a special hidden unit able to store inputs over many time steps and act as long-time memory. This memory cell is connected to itself at the next timestep, thus copying its own real-valued state and accumulating the external signal. Another hidden unit, the *forget gate*, which learns to decide when to clear the content of the memory, is used as a multiplicative gate of this self-connection.

For each input vector, the recurrent layer performs the following computations:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i \mathbf{x}_t + \mathbf{b}^i + \mathbf{U}_h^i \mathbf{h}_{t-1} + \mathbf{b}_h^i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}^f \mathbf{x}_t + \mathbf{b}^f + \mathbf{U}_h^f \mathbf{h}_{t-1} + \mathbf{b}_h^f) \\
\mathbf{g}_t &= \tanh(\mathbf{W}^g \mathbf{x}_t + \mathbf{b}^g + \mathbf{U}_h^g \mathbf{h}_{t-1} + \mathbf{b}_h^g) \\
\mathbf{o}_t &= \sigma(\mathbf{W}^o \mathbf{x}_t + \mathbf{b}^o + \mathbf{U}_h^o \mathbf{h}_{t-1} + \mathbf{b}_h^o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{q}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{2.31}
$$

Where $\mathbf{h}_t$, $\mathbf{c}_t$ and $\mathbf{x}_t$ are respectively the hidden state, the cell state and the input state at time t, $\mathbf{h}_{t-1}$ is the hidden state at time *t-1*; $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{g}_t$ and $\mathbf{o}_t$ are the input, forget, cell and output gates respectively. σ is the sigmoid activation function and $\odot$ is the Hadamard product (element-wise product). The overall process followed in Eq 2.31 can be visualized in Fig. 2.9, where a single LSTM cell schematic is shown. For a more detailed explanation about LSTM refer to [47, 48].

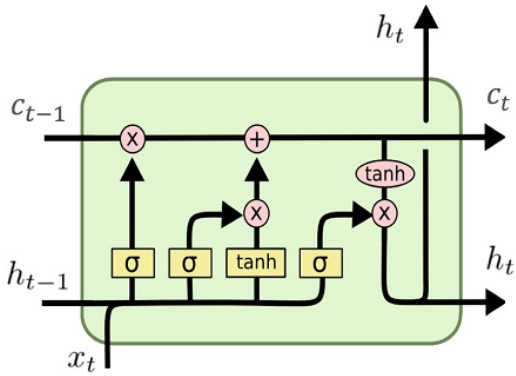


Figure 2.9: LSTM scheme. *Source:* `https://commons.wikimedia.org/wiki/File:LSTM.png`

For learning very complex functions, various RNNs can be stacked together to form deep RNNs. Due to the temporal dimension, training only one RNN layer can already be computationally expensive. For this reason, deep RNNs remain rather shallow in practice when compared to feedforward neural networks, which very often have a great number of layers.

2.2.3.3 Back-propagation training process

Every ANN needs to be trained in order to accomplish its objectives; the training process allows to find the best values for neurons' weights and biases. In order to quantify how well the network is working a *cost* function, also called *loss* or *objective* function, has to be selected. Typical loss functions

are the Mean Squared Error (MSE), L_1 *loss* or *Cross Entropy*. In general, the training algorithm aims to minimize the cost as a function of weights and biases; meaning that the algorithm is doing well if it finds weights and biases capable of reducing the value of the loss function.

For succeeding in the convergence, the choice of the loss function is fundamental. It is directly related to the activation function used in the output layer of the network; indeed, if the network output layer represents the framing of the prediction problem, the loss function is the way of calculating the error of that given framing. Some significant examples are listed below [49]:

- **Regression Problem**: it is a problem when a real-value quantity has to be predicted. Typically the output layer is a single node with a linear activation function. In this case, the best choice for the loss is the MSE.
- **Binary Classification Problem**: it is a problem when the input must be classified into one or two classes. In this case, usually, the activation function of the output layer is a sigmoid. The best choice is the Cross Entropy (CE), also named Logarithmic loss.
- **Multi-Class Classification Problem**: similar to the previous one, but with more than two classes. The usual output layer configuration is one node for every class using a softmax activation function. Also, in this case, the best choice falls back on the Cross Entropy.

MSE and the Cross-Entropy for a binary problem loss function can be defined as follows:

$$\text{MSE} = \frac{1}{n}\sum_{j=1}^{n}(y_j - \tilde{y}_j)^2 \tag{2.32}$$

$$\text{CE} = -(y\log(\tilde{y}) + (1-y)\log(1-\tilde{y})) \tag{2.33}$$

where y is the actual value and $\tilde{y}$ is the predicted one. If the problem is multi-class classification, in the CE equation, a separate loss for each class label is computed and then summed together.

$$\text{CE} = -\sum_{j=1}^{N} y_j \log(\tilde{y}_j) \tag{2.34}$$

The most common algorithm used to perform this kind of training is the *gradient descent* algorithm. More complex algorithms exist: one of the most used is the Adam optimizer, born by the combination of other two extensions of stochastic gradient descent, which are the Adaptive Gradient Algorithm

(AdaGrad) and the Root Mean Square Propagation (RMSProp) [50]. The weights and the biases are updated by computing the derivative of the loss function with respect to weights and biases themselves as in the following equations:

$$w_{ij}^k = w_{ij}^k + \Delta w_{ij}^k = w_{ij}^k - \eta \frac{\partial L}{\partial w_{ij}^k} \tag{2.35}$$

$$b_i^k = b_i^k - \Delta b_i^k = b_i^k - \eta \frac{\partial L}{\partial b_i^k} \tag{2.36}$$

Where η is the learning rate and L is the loss function. Since the learning rate scales the derivative, it is very important for the convergence of the gradient descent algorithm: if the L value is too small, the learning converges too slowly, if it is too high it could generate oscillations and instability problems. There are some methods, called adaptive methods, in which the learning rate is modified according to the behaviour of the loss function.

To sum up, each training step is divided into two phases: the *forward* and the *backward* phase. During the first, the input is propagated into the entire neural network, generating the final output, as shown in Eq. 2.28. Then, in the backward phase, the loss is computed and propagated again but in the backward direction, to compute all the gradients. At the end, both weights and biases are updated. Ideally, this process could last forever, therefore, it is very important to decide when to interrupt it. Indeed, an early stopping can lead to a non-complete input-output mapping; while if the training stops too late, the results obtained can be overfitted and therefore the network could lack some capabilities. The common way to deal with this problem is to interrupt the training when the gradient vector falls below a fixed tolerance value: it means that the gradient descent has almost reached the minimum, whether it is global or local. In problems where data are known, like in supervision learning, the *validation* criterion can be used. The training processes can be distinguished also into instance-incremental training process or batch-incremental training process. The former trains the network at each time step, and the latter trains the network only after a defined number of steps. A hybrid process is the so-called *mini-batches* training process [51].

2.2.4 DRL Algorithms

Almost the totality of DRL algorithms can be divided into two main categories: the *policy-based* methods and the *value-based* methods. On one hand, the approach consists of searching for the policy that behaves correctly in a specific environment; on the other hand, instead, of trying to value the utility of taking a particular action at a specific state. Sometimes, the literature tends

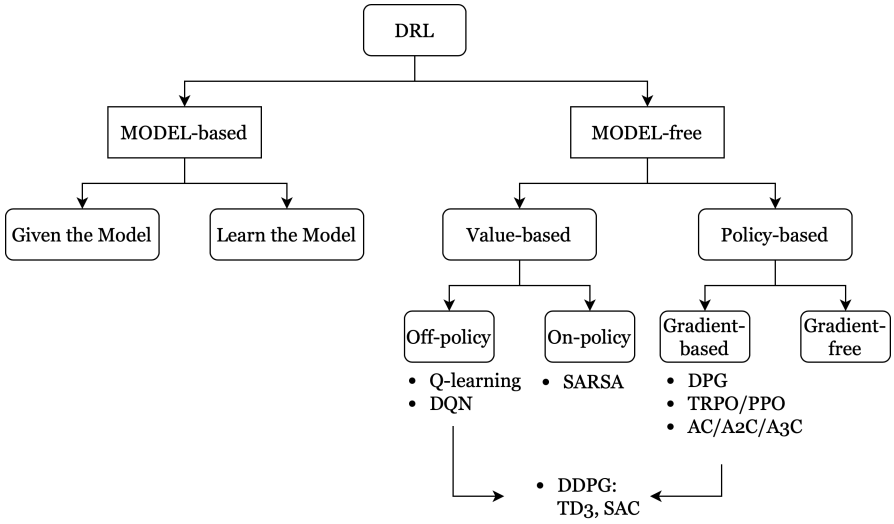


Figure 2.10: High-level (and not exhaustive) taxonomy of DRL algorithms [52].

to identify *model-based* approaches as the third cluster of this classification, but, here, it has been reckoned at a higher level. An additional distinction in reinforcement learning is *on-policy* and *off-policy*. The former methods attempt to evaluate or improve the policy that is used to make decisions during training; whereas the latter methods evaluate or improve a policy different from the one used to generate the data, i.e., the experience. It is not so easy to strictly categorize all the DRL algorithms, especially because their spectrum is very wide and fast growing. An high-level categorization[1] is shown in Fig. 2.10.

In the next two subsections, the two algorithms treated in the following chapters of the thesis will be illustrated. In particular, the A2C is one of the baseline algorithms from which the PPO is developed. A2C performance have been deeply analysed in [26].

2.2.4.1 A2C

A2C methods represent the improvement of the basic Actor-Critic algorithm. In general, Actor-Critic methods merge the advantages of value-based methods (i.e. DQN) and policy-based methods (i.e. Policy gradient) [24]. They split the agent model into two parts, one that computes the action and the other that

[1]OpenAI Spinning Up documentation: `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`

computes the action-value function[1]. For this reason, the two components are then called respectively:

- *Actor*, which takes as input the state and generates the *perceived* best action as output, thus, controlling the agent behaviour.
- *Critic*, which, instead, evaluates the action taken by the actor computing the corresponding action-value function.

Both actor and critic are function approximators, embodied by ANN. The training process of the two networks is performed separately and exploits the gradient ascent for parameter update. During the training, the actor learns to produce better actions while the critic learns to evaluate better those actions. The update may occur both at the end of each time step (TD-0) or at the end of an episode (TD-n). Among actor-critic methods, the A2C shows good performance on continuous control problems. The main feature of the A2C algorithm is the employment of an *advantage* value function $A(s_t, a_t)$, which is created by subtracting the value function $V(s_t, a_t)$ from the action-value function $Q(s_t, a_t)$, as depicted in Eq. 2.37. All these quantities depend on state s and action a at the time step t.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t, a_t) \tag{2.37}$$

The advantage function represents how better an action is compared to all the other possible actions in a given state; it is different from the value function which instead captures how good it is to be in a particular state. As already introduced in Section 2.2.2, exploiting the Bellman equation [39] for $Q(s, a)$, it is possible to write the advantage function $A(s, a)$ only in dependence of the value-function $V(s, a)$, the reward r, and a discount factor, γ.

$$A(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})] - V_v(s_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t) \tag{2.38}$$

In *A2C*, the critic network is used to approximate the advantage value function in Eq. 2.38, meaning that, instead of learning the Q-values, the critic network learns the A-values; thus, the evaluation of an action is not only based on how good the action is but also based on how better it can be. The direct consequence is the reduction of the high variance of the policy at each update and, therefore, the stabilization of the learning model. If the behaviour policy is equal to the updated policy, the method is called *on-policy* learning (i.e. TD-0). Instead, the learning is *off-policy*, whenever the agent learns when the

[1] S. Karagiannakos, "The idea behind Actor-Critics and how A2C and A3C improve them", Nov 17, 2018, `https://theaisummer.com/Actor_critics/`

episode is concluded, and, therefore, the update and behaviour policies are different (TD-n). In Algorithm 1, the pseudo-code for the A2C-TD-n algorithm is presented.

Algorithm 1 A2C Algorithm - TD-n

Initialize actor network: $\rightarrow \mathcal{A}_0 = \mathcal{A}(s, a \mid \theta^- = \theta_0)$
Initialize critic network: $\rightarrow \mathcal{C}_0 = \mathcal{C}(s, a \mid \theta^- = \theta_0)$
for episode = 1,N **do**
    Initialize sequence s_0
    while not done **do**
        $a_i = \text{argmax}_a \, \mathcal{A}_k(s_i, a \mid \theta)$
        compute reward r_i and new state s'_i
        save reward r_i in $\underline{\mathbf{r}}$ vector
        compute $V_i(s)$ from critic
        save $V_i(s)$ in $\underline{\mathbf{V}}$ vector
    end while
    compute in reverse with critic $\underline{\mathbf{Q}} = \underline{\mathbf{r}} + \gamma \underline{\mathbf{V}}$
    train network: $\mathcal{A}_{k+1} = \text{train}()$, $\mathcal{C}_{k+1} = \text{train}()$
    update parameters θ
end for

2.2.4.2 PPO

As outlined in Fig. 2.10, PPO is a policy-gradient method, belonging to the Actor-Critic family [53]. It outclasses most of the other DRL algorithms in many typical benchmark problems, because of its improved training stability. It builds up from the Trust Region Policy Optimization (TRPO) method [54], retaining its reliability and data efficiency, but handling the loss function in a much simpler and well-planned fashion. Starting from the TRPO loss function, which exploits the probability ratio between the policy π_w at two subsequent timesteps (as in Eq. 2.39), PPO increases training robustness by clipping the objective function and limiting the possible update so that the policy does not change drastically. The simple expression for the PPO loss function $L^{CLIP}(w)$ is reported in Eq. 2.40.

$$p_k(w) = \frac{\pi_w\left(a_k|s_k\right)}{\pi_w\left(a_{k-1}|s_{k-1}\right)} \tag{2.39}$$

$$L^{CLIP}(w) = \hat{\mathbb{E}}_k\left[\, min\left(p_k(w)\,,\, clip\left(p_k(w), 1-\epsilon, 1+\epsilon\right)\right)\right] A_k \tag{2.40}$$

Where w refers to the network's parameters (i.e. their weights and biases); the parameter ϵ indicates the clipping factor, A_k is the advantage function, retained from the A2C formulation, as in eq. 2.38).

In Eq. 2.40, the clipping function *clip* limits the probability ratio to be inside the range defined by $1+\epsilon$ and $1-\epsilon$. This means that thanks to the clipping objective function, multiple epochs of gradient descent can be run on the sample data without causing destructively large policy updates and squeezing every ounce of information it can learn from.

Moreover, a common practice in PPO algorithms is the addition of an entropy regularization term multiplying the state s_k to the clipping objective function, as in Eq. 2.41, to ensure a sufficient exploration level during training:

$$L^{PPO}(w) = L^{CLIP}(w) + c_2 S(\pi_w) s_k \tag{2.41}$$

Where $S(\pi_w)$ is the entropy bonus term, which is the function of the current policy, and c_2 is a scalar multiplying factor that determines the influence of the entropy term on the overall loss function. If the actor network is trained from the clipping objective function, the critic network is, instead trained by means of optimizing a simple MSE objective function, defined in Eq. 2.42.

$$L_{critic} = \sum_{i=1}^{N} \left(V(s_k^i) - \left[\sum_{j=k}^{T} \gamma^{j-k} r(s_j^i, a_j^i) \right] \right)^2 \tag{2.42}$$

Where the subscript k stands for the considered time-step instant; while the superscript i stands for the current batch used for the loss function computation. Regarding the practical implementation of the method, a number of hyperparameters need to be introduced and detailed. For an in-depth description of all the parameters refer to the original PPO paper [16]. The most important ones are listed below:

- the discount factor γ, already discussed in Section 2.2.2, ruling how farsighted is the agent;
- the Generative Adversarial Estimator λ also contributes to reward shaping;
- the clipping factor ϵ corresponds to the acceptable threshold of divergence between the old and new policies during gradient descent updating. Setting this value to a small number will result in more stable updates, but will also slow the training process;
- the entropy coefficient β acts as a regularizer and prevents premature convergence which in turn may prevent sufficient exploration;
- the batch size corresponds to how many experience time-steps are used for each gradient descent update;

- the buffer size corresponds to how many experiences should be collected before gradient descent is performed on all of them. This should be a multiple of the batch size, otherwise a batch is truncated and may poorly affect the optimization step. Typically larger buffer sizes correspond to more stable training updates;
- the number of epochs is the number of passes through the experience buffer during gradient descent. The larger the batch size, the larger it is acceptable to make this. Decreasing the number of epochs will ensure more stable updates, at the cost of slower learning.

Algorithm 2 Proximal Policy Optimization - PPO-Clip

1: Input: initialization policy parameters θ_0, initialization value parameters ψ_0
2: Initialize batch
3: **for** k = 0, 1, 2, ... **do**
4: **while** batch-step $b_i \leq$ batch-size **do**
5: Collect set of trajectories $\mathcal{T}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment
6: Compute Rewards $\hat{\mathcal{R}}_k$
7: Compute Advantages $\hat{\mathcal{A}}_k$ based on the current value function $\hat{V}_{\psi_k}$

$$A_\pi(x_i, u_i) = \Big[\sum_{j=i}^{T} \gamma^{j-i} r(x_j, u_j)\Big] - V_\psi(x_i)$$

8: **end while**
9: Compute the probability ratio $p_k(\theta_k)$
10: Update the policy by maximizing the clipping objective function via stochastic gradient descend (ADAM optimizer):

$$L_{\text{CLIP}}(\theta) = \mathbb{E}_{p(\tau)}[\min[p_k(\theta), \text{clip}(p_k(\theta), 1-\epsilon, 1+\epsilon)]A_\pi(u_k, x_k)]$$

11: Update value function by regression on mean-squared error:

$$L_V = \sum_{i=1}^{N} \Big(V_\psi(x_k^i) - \Big[\sum_{j=k}^{T} \gamma^{j-k} r(x_j^i, u_j^i)\Big]\Big)^2$$

12: **end for**

In conclusion, the algorithm usually loops until it has collected a certain batch size of data, obtained through the interaction of the agent with the environment. At each step, it stores a set of observations, actions, rewards and advantages values. Once it retrieves a number of transitions specified by the batch dimension of the experience buffer, the probability ratio and consequently

the loss functions for both the actor and the critic are computed as in Eq. 2.39, Eq. 2.40 and Eq. 2.42. With these losses, the two neural networks are then updated via back-propagation with an *Adam* optimizer. This way the networks adjust their parameters to better fit the problem objectives. The pseudo-code of PPO algorithm is outlined in Algorithm 2.

All these concepts will be later resumed in Section 3.4, when a brief hyperparameter analysis will be discussed.

2.3 Image Processing

The main problem treated in this thesis is designed to the shape (also called *map* in the following chapters) of an unknown and uncooperative space object such as debris, or a dismissed satellite (in this case, it may partially or completely be known).

Since the spacecraft will be commanded by a DRL trained agents, finding some metrics to quantify or measure the *mapping* level becomes of great importance. The most valuable thing is, therefore, to select some general criteria that define how good is the image of an object taken by a space camera. Since the image reconstruction does not fall within the objective of this thesis, these criteria may not be directly related to a specific technique but may be generally purposed, if possible.

In the following section, a very brief overview of the nowadays state-of-the-art methodologies for 3D image reconstruction on objects is described, and afterwards, the requirements needed for the reward model definition are derived.

2.3.1 Techniques Overview

In general, three-dimensional object or environment reconstruction is a long-standing ill-posed problem, which has been explored by different research branches such as computer vision, computer graphics, and in the last decade machine learning, especially with the increasing exploitation of CNN [55]. Generally, several techniques are based on encoder-decoder networks for the 3D shape reconstruction of generic objects based on either single or multiple RGB images for volumetric [56] or surface-based representations [57].

All these techniques are in some way general-purpose, even if in the last years also analyses directly applied to space objects have increased in number. For example, in [58] a novel approach to recovering the object structure model from multi-view images captured by a visible sensor is studied. The methodology is still based on the widely used structure from motion (SFM) methods and patch-based multi-view stereo (PMVS) algorithm. Another process is explored in [59] for shape reconstruction of non-cooperative space objects adapting

instant NGP-accelerated NeRF and D-NeRF (mainly designed for static object modelling) to the mapping problem. This work also analysed how the lighting and illumination conditions strongly influence the potential reconstruction.

Concerning instead, small bodies (i.e., asteroids, comets) geometry reconstruction from images, and different techniques have been studied through the years. High-fidelity topography is generally possible only when the images are taken very close to the object. Stereophotoclinometry (SPC) and Stereophotogrammetry (SPG) are state-of-the-art methods that guarantee high-resolution models given a sufficient number of observations. In particular, SPC is an evolution of the shape-from-shading approach and links photometry to stereoscopy. In other words, from *maplets* initialization and images with navigation data, brightness is extracted to create a photometric model. Then, slopes, defined as the inclination of the *maplets* with respect to the ideal plane in which the photo lays, and albedo are estimated from the photometric model [60]. The overall process is based on a camera model defined by stereoscopy.

In [61], some high-level requirements have been derived to obtain high-quality SPC models.

- A minimum of three images per face (typically > 40).
- Emission angle, defined as the angle between the camera direction and the normal direction to the face, should be maintained around $\sim 45°$ (optimal range 35°-48°, limit range 5°-60°).
- Incidence angle, defined as the angle between the Sun direction and the normal direction to the face, should be maintained around $\sim 45°$ (optimal range 30°-50°, limit range 0°-70°).
- Variation in illumination conditions (optimal range 40°-90°, limit range 10°-120°). Variation in illumination conditions is a direct function of the variation of the Sun incidence angle.

Such requirements, even if defined for a small body reconstruction technique, so not properly shaped for a space object, are still reputed enough adequate for the definition of the reward model, which includes camera and illumination criteria for defining the mapping level. Similar models have been defined in [14]. In the *optimal* range, the best image possible is expected. In the *limit* range, instead, a lower quality image is obtained, still usable for the image process. In the reward mode l the emission angle will be called *camera incidence angle* and the incidence angle will be named as *Sun incidence angle*. For the reward model definition, the *optimal* ranges will be wider than the ones stated here, while the *limit* ranges will be maintained equally. This design choice is finalized to accelerate the learning through higher rewards for a wider space of possible states.

2.3.2 Navigation with Images

In Chapter 5 a very brief explanation of image processing and navigation filters will be treated, focusing the attention on the tools that will be helpful for the development of the GNC pipeline. Such kinds of topics exceed the framework of the thesis's main subjects and therefore, an extensive state-of-the-art analysis is unnecessary. Nevertheless, it is still worth spending some words on the variety of image-based navigation techniques for space applications, especially because they are becoming more and more important, particularly for rendezvous, docking, planetary landing, or mapping missions. This kind of navigation may exploit visible or thermal image pieces of information, which are used to determine the spacecraft's position and orientation in space. The nowadays research on this topic is clustering its attention mostly towards feature matching, image registration, stereo vision, optical flow and pattern recognition with machine learning techniques [6]. In particular, feature matching methodologies involve the identification, matching and comparison of distinctive landmarks or features in captured images with known landmarks or maps to directly retrieve their position and orientation relative to those landmarks. Image registration techniques, instead, align different images of the same scene to create a composite or map. Also, in this case, the comparison between the registered images and known maps or reference images makes possible the estimation of the spacecraft's state. Both the previous two cases do not require multiple images from different cameras; if, instead, more than one camera is present on-board, a stereo-vision-based technique may be exploited to acquire multiple images from different perspectives, and, therefore, evaluate relative information by analysing the disparities between these different images. If no known maps or feature points are available, optical flow analysis methods can be exploited to track the movement of the visual features between consecutive image frames [9, 8]. At last, utilizing pattern recognition algorithms and machine learning techniques, the spacecraft's ability to identify and navigate on images can be enhanced, helping the GNC in distinguishing objects, understanding environments, and predicting movements [62].

In general, all these techniques have been introduced for visible image information. Nevertheless, since objects in space emit different levels of heat, also thermal information may be utilized for navigation purposes. Analysing thermal patterns can help in identifying objects, understanding their relative positions, and navigating accordingly [63]. In conclusion, this very high level of information will be very useful in understanding the rationals behind the selection of the image processing and navigation methods about the GNC algorithm design.

2.4 Overview on available Software Tools

It is worth spending some words about the available machine learning frameworks and tools. Indeed, nowadays, Deep Reinforcement Learning is one of the hottest topics in the Data Science community, and thanks to its fast development, the demand for easy-to-understand and convenient-to-use RL tools is rapidly growing. In recent years, plenty of RL libraries have been developed and designed to have all the necessary tools to implement and test RL models.

Nowadays, the most common and easy environment in which developing an DRL problem is Python, and the most popular machine learning frameworks designed for it are:

- **TensorFlow**, open-source framework developed by Google Brain. It has a rich ecosystem with tools like TensorBoard for visualization, extensive documentation and a large user community. Moreover, it is greatly suited for integration with hardware accelerators, like GPUs and TPUs.

- **PyTorch**, lightweight open-source framework developed by Facebook. It is known for great dynamic computation graphs that allow easy debugging and flexible model architectures. It has strong support for custom layers and loss functions and an intuitive interface with a thriving community and rich ecosystem of libraries.

- **Keras**, an open-source framework, developed on top of TensorFlow, shaped to be efficient with high level neural networks.

Besides these most popular and general-purposed tools, there are several other tools more complex or suited for specific networks or problems, that cannot be completely open-source: i.e., Apache MXNet, Caffe (for C++), Theano (not open), Shogun, Scikit Learn (better for unsupervised learning), and several others.

In Table 2.1 the three frameworks are compared on different useful aspects.

Aspect	TensorFlow	PyTorch	Keras
Usability	Moderate	Easy	Very Easy
Flexibility	High	High	Moderate
Community Support	Strong	Growing	Strong
Popularity	Very Popular	Popular	Popular
Dynamic Computation	Static	Dynamic	N/A
Ecosystem	Extensive	Growing	Growing
Production Readiness	Excellent	Good	Good
Integration	HW Accelerators	Limited HW support	Backend dependent

Table 2.1: Comparison of the most popular deep learning frameworks

For this work, opting for an easier and more dynamic approach, PyTorch has been selected. All the simulations have been run on Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz processor, with 16Gb RAM available. No GPU has been used. Since these specifications limit strongly the performance, PyTorch was the optimal choice, thanks to its lightweight.

CHAPTER 3

Problem Architecture

> Certainty of death. Small chance of success. What are we waiting for?
>
> — Gimli, *The Fellowship of the Ring*

This thesis proposes an innovative decision-making process to autonomously plan the pseudo-optimal guidance around the uncooperative space objects through the exploitation of Deep Reinforcement Learning. The definition of the scenario starts with is coupled with a pre-processing phase, in which information coming from the external environment, sensors and object conditions are elaborated to estimate the state. The state is then fed to the autonomous guidance agent which crafts the control policy to maximize the reward, affecting the environment and all the other information providers.

In this work, three main scenarios have been studied:

- **General-purposed**: a general chaser-target setting, in which the target object is defined as a general-purpose object shaped as a parallelepiped. This case is studied in Section 4.2 to assess the baseline of the analysis and

is used to study and define the neural network architectures, the hyper-parameters, the reward robustness and the initial condition sensitivity. For this case, Visible Imaging Sensor (VIS) camera system is assumed for navigation and map reconstruction.

- **VESPA** (Vega Secondary Payload Adapter): in this setting a VIS and Thermal Infrared (TIR) camera system is considered as proposed in [63] in order to avoid illumination condition problem proper of a VIS-only vision-based system and, therefore, to relax the reward model. This case will be studied in the continuous action space in Section 4.3 and exploited for the benchmark analysis in Section 4.6 in comparison to the guidance control of the e.Inspector mission [64].

- **TANGO**: in which the chaser is equipped with a VIS-based navigation system to fly around the TANGO satellite, part of the Swedish PRISMA mission, developed as a technology demonstrator for sensor technologies and guidance/navigation for rendezvous and formation flying in space launched in 2010 [65]. This scenario will be deeply analysed in a full AI-based GNC pipeline in Chapter 5.

In this chapter, the justification and explanation of all the different aspects needed for the definition of each of the three scenarios will be presented in terms of environmental formulation (Section 3.1) and modelling (Section 3.2), policy and value networks' architectures settings (Section 3.3), and, finally, DRL algorithm hyper-parameters analysis.

3.1 Environmental Model

The environment is the representation of the entire dynamical framework in which the agent must learn how to act properly. From it, the three main models composing a typical DRL problem branch out: state, action and reward model. The state space is part of the environment and it is influenced also by the action space; the combination of the action and environment model generates the reward state. Their most important features are:

- The chaser-target relative dynamics, computed as explained in Section 2.1.

- The portion of the object that is observable by the chaser's camera depends on the relative target orientation.

- The Sun-chaser relative dynamics that derive from the dynamics of the object in the main attractor system (i.e. Earth system), and on the relative position between the Sun and attractor, that most accurately depends on the ephemeris model. The Sun's relative position is computed as briefly explained in Section 3.2.

While the Sun relative dynamic is not affected by anything during the simulation, since it depends only on the initial condition defined, the chaser-target combined dynamics is directly affected by the choices made by the agent. Hence, at each time step, knowing the previous environment condition and the selected action, it is possible to compute the new environment condition, from which the new state is extrapolated to feed the agent learning neural network.

If on one hand, the definition of the initial conditions is essential for the correct behaviour of all the components of the scenario, on the other hand, in order to define the simulation settings, it is also very important to understand which the limits of the environment are, and, therefore the simulation ending conditions. In Section 4.1, the setting of the initial fixed or random conditions will be discussed. Concerning the final conditions, the features that could influence the termination of a single episode can be more than one:

- The absolute distance between spacecraft and object: if it is less than a minimum distance the spacecraft is believed to have impacted against the object, thus the mission is compromised; if the distance grows higher than a maximum one then the spacecraft has moved so far away to get usable measurements: also, in this case, the mission is compromised and the simulation can be terminated.

- The time of the simulation cannot be endless, but it may be restricted to a specific time window, in which the agents must have the capability to accomplish its mission.

- Another parameter that may affect the episode conclusion could be the complete achievement of the mission requested. In this case, if the agent is capable of reconstructing the totality of the object's shape collecting the required amount of photos then the simulation may be considered succeeded and, therefore, concluded.

Setting the problem in this way, no particular final conditions are requested, i.e. specific final position requested for the spacecraft. This determines a lack of continuity with possible other missions consecutive to the one completed by the spacecraft. Introducing a requirement like that in this kind of formulation is quite hard, but being aware of this drawback is useful to leave nothing to chance.

In the next subsections, a detailed definition of the proposed models for the autonomous decision process will be presented. The general environment space is characterized by:

- Action and state spaces: which define the action among whom the agent has to choose at each time step basing the choice on the state variables that are accessible to its knowledge.

- Reward model: it defines the reward function that the agent has to maximize. Hence, it collects all the good and bad parameters that affect the agent success

3.1.1 State Space Model

The state space model shall synthesize all the minimal information needed by the agent to have a partial or complete overview of the environment in order to be able to learn from experience. As introduced in the previous chapters, it is impossible for an agent representing a spacecraft to have access to all the information characterizing a particular space environment. Therefore, it has to deal with a reduced state space with respect to the one that completely defines the space, thus the formulation of the POMDP. Since the input vector should be tailored in such a way as to contain only essential information for the decision-making process, to build a policy capable of selecting the appropriate action in every condition the agent may find itself. In this case, the variables are the ones that properly identify the close proximity scenario and facilitate the agent learning. Moreover, a key factor to be considered is the possibility of estimating these quantities employing on-board instruments, which is of fundamental importance for an autonomous spacecraft. Therefore, the input to the agent has to be defined as similar as possible to a realistic bunch of information to which the spacecraft can actually get access, the overall state-space is designed to include the relative state between the chaser and the target, both in terms of position, velocity and orientation. This means that information on the chaser-target absolute position with respect to the Earth, or the Sun orientation, the residual mass or the map level is not available to the agent.

In many cases, it is useful to use statistical or mean quantities to reduce the number of states mapped by the agent despite the lack of accuracy of the result. To ease understanding of how the map is built or how the relative position between the two is, the history of past action may be helpful in the state space. Nevertheless, despite the potential benefit, at first, both historical and statistical information were excluded, reducing the observation space but facilitating the achievement of the final result in terms of computation lightness and learning time. Historical information will be implicitly added with the exploitation of RNN architecture.

Therefore, the state space can be defined as in Eq. 3.1:

$$\boldsymbol{\mathcal{S}} = \begin{Bmatrix} \mathbf{r} \\ \mathbf{v} \\ \boldsymbol{\alpha} \\ \dot{\boldsymbol{\alpha}} \end{Bmatrix} = \begin{Bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \alpha_x \\ \alpha_y \\ \alpha_z \\ \dot{\alpha}_x \\ \dot{\alpha}_y \\ \dot{\alpha}_z \end{Bmatrix} \tag{3.1}$$

Where $\mathbf{r}$ and $\mathbf{v}$ are the chaser-target relative position and velocity, and $\boldsymbol{\alpha}$ and $\dot{\boldsymbol{\alpha}}$ are the angular position and velocity of the target, characterizing the relative orientation between the two objects.

Obviously, in practice, all the input data are derived from sensor measurements, which inherently introduce errors. This uncertainty is studied in Section 4.4, where the sensitivity analysis tests have been performed. Further analysis can be developed starting from this initial problem, like evaluating the consumed propellant mass. Indeed, if part of the objective requires the minimum propellant consumption, the mass of the chaser could be added to the state in order to track the consumption through the evolution of this parameter [66]. Nevertheless, in several case studies, it will be observable how the agent usually learns to take the minimum number of actions without any explicit constraint on the propellant consumption. Thus, this augmented state most of the time is unnecessary.

3.1.2 Action Space Model

The action space is the second pillar on which RL is built, representing the space of all the possible decisions the agent could take at each *timestep*. In this work, the agent can interact with the surrounding environment by means of the acceleration vector a, representative of the control action, that directly affects the equations of motion as expressed in Eq. 2.12. This kind of action control can easily represent a thrust vector coming from the thruster and therefore affects the dynamics as follows:

$$\begin{cases} \ddot{x} = \left(\dfrac{2\mu}{r^3} + \dfrac{h^2}{r^4}\right) x - \dfrac{2(\mathbf{v}\cdot\mathbf{r})h}{r^4} y + \dfrac{2h}{r^2}\dot{y} + a_x \\ \ddot{y} = \left(\dfrac{h^2}{r^4} - \dfrac{\mu}{r^3}\right) y + \dfrac{2(\mathbf{v}\cdot\mathbf{r})h}{r^4} x - \dfrac{2h}{r^2}\dot{x} + a_y \\ \ddot{z} = -\dfrac{\mu}{r^3} z + a_z \end{cases} \tag{3.2}$$

where, a_x, a_y, a_z are the three components of the control vector commanded by the spacecraft.

Concerning the action space, two different approaches can be adopted, *discrete* and *continuous*, as defined in the two following subsections. This substantial difference is strongly important also for the algorithm selection. Indeed, if on one hand A2C is more suited for discrete state-action space problems, PPO can deal the same with both discrete and continuous state-action space.

3.1.2.1 Discrete

A discrete action space is defined by a set of predefined options, which in this case means having specific thrust impulses fixed both in direction and magnitude, as in Eq. 3.3.

$$\mathcal{A} = [\, +T_x,\ -T_x,\ +T_y,\ -T_y,\ +T_z,\ -T_z,\ 0\,] \tag{3.3}$$

where, for example, $+T_x$ defines the thrust action aligned with the +x direction of the reference frame. Therefore, the agent will be allowed to select only one of the options within $\mathcal{A}$. This kind of approach requires that also the level of acceleration given by the thruster must be previously defined, thus leading to two different design choices:

- **Fixed acceleration level**, e.g. 0.001 m/s^2; in this way, the need for knowledge of the mass of the system is overridden and the actual level of thrust given by the thruster decreases with the reduction of the propellant and mass. Therefore, both of them become redundant and superfluous information. Even if less straightforward, this option reduces the complexity of the environment, which otherwise would have need also the integration of the Tsiolkovski equation for the acceleration estimation.
- **Fixed thrust level**, e.g. 0.01 N; unlike before, this option is way more realistic and coherent, considering how spacecraft's thrusters work, but, as already underlined, it would require an additional brick in the complex definition of the environment.

Therefore, to ease the computations, the first option is selected when working with discrete action space, meaning that if the agent selects the action $\mathcal{A}[2] = -T_x$, the corresponding control action becomes $[0,\ -0.001,\ 0]\ m/s^2$. The coherence of this kind of approach with orbital control actuators has already been assessed in Sec. 7.3 of [26].

3.1.2.2 Continuous

The *continuous* action space approach, instead, sees the control action becoming a tridimensional vector, ideally free to point towards the entire 3D space, without being constrained by some specific direction. Moreover, since it is continuous, the magnitude of the thrust vector can vary inside the limits specified by the propulsion system on-board. To define the action space, an electric propulsion system with a single thruster [67] is employed. The most notable attribute affecting the formulation is the maximum thrust and the minimum impulse bit (MIB), since they define the range inside which the decision-making policy can select the magnitude of the action.

Another important parameter for the action model is the control interval Δt, which defines the time elapsing between one control action and the other. The setting of these parameters entails a trade-off between fidelity of the control frequency and computational burden.

As in the discrete case, the design choice between acceleration or thrust arises. For the same reasons explained above, the more useful approach is to have the agent directly selecting the acceleration vector. Therefore, the control action will be something as $[0.01,\ -0.026,\ 0.008]\ m/s^2$. This approach will be extensively discussed in Section 4.3.

3.1.3 Reward Model

In RL, an agent interacts with an environment by taking actions, and in response, it receives feedback in the form of rewards. The agent's objective is to learn a policy that, over time, maximizes the expected cumulative reward. The reward model provides a numerical value that represents the immediate desirability of the state-action pairs, informing the agent about the consequences of its actions. Therefore, one of the most significant and delicate parts of this architecture is the definition of the reward model since the learning policy depends mostly on it. Therefore, it must be defined carefully and in such a way that the right importance is given to all the objectives. As intuitive, the more complex the problem is, the more complex the reward model is going to be. In this particular analysis, different goals share the definition of the reward model.

This particular problem is characterized by different mandatory or *nice-to-have* objectives:

- **Safe trajectory**: that means avoiding some regions of the space that are considered dangerous and therefore bringing to the end of the episode simulation.
- **Time window**: the agent has to learn to compute all the needed operations in the correct and fixed time window, neither too short nor too long.
- **Target object map reconstruction**: this goal depends on the adopted mapping technique. Here, it is defined to ease SPC, as introduced in Section 2.3; nevertheless, if a different shape reconstruction technique is considered, dissimilar scores can be defined without changing the main intent of the procedure.
- **Good quality images**: as explained in Section 3.2, the object is composed of a certain number of faces; therefore, for each of them the goodness of their exposition to the Sun and the spacecraft's camera can be defined.
- **Thrust level**: limiting the use of propellant for trajectory control.

All of these objectives can be divided into different scores that can be assembled together in the total reward score, which can change in order to have simpler or more complex rewards or modified with different levels of importance for each of the objectives. It is important to note how could happen that some objectives may be in contrast and need to be balanced: a planner that maximizes the coverage would explore new areas, while a planner that maximizes the map quality would spend more time on the same areas.

In the next list, all the scores that will be used in the different kinds of simulations and analysis are defined and explained:

- *distance score*: in the case of proximity operations, a general and intuitive idea is that the chaser spacecraft shall not crash onto the target, nor escape far away from it. This constraint is formulated by adopting a lower and upper limit in terms of relative distance between the two objects. In this way, it is intrinsically introduced safety in the operations, by incentivizing the agent to avoid dangerous regions of space, in which the mission would completely fail. Note that this bounded region represents also a possible terminal state for the agent: if the lower or the upper limit is overcome, then the episode ends and the agent is given a negative reward signal. The associated mathematical expression and score are reported in Eq. 3.4:

$$r_d = \begin{cases} -100 & \text{if } d \leq D_{min} \text{ or } d \geq D_{max} \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

where $d = \|p\|$ is the distance between chaser and target, with $D_{min} = 50m$ and $D_{max} = 500m$.

- *incidence angle score*: regarding the main goal of the presented work, the agent should maximize a reward function that enables it to better map the target. This is strictly connected to the adopted mapping technique, which requires asserting some specific conditions in terms of incidence angle and number of quality images per mesh face, as discussed before. At each timestep, the agent keeps track of the target rotation and re-computes the normal direction for each of the mesh faces. First, a screening of the faces that are in the field of view of the spacecraft's cameras is performed. Then, the angle, ε, between each of the normal directions of the faces in visibility and the camera vector, assumed as continuously pointing the target centre, is calculated. A score, reported in Eq. 3.5, is formulated on the base of the visibility model defined in Section 3.2:

$$r_\varepsilon = \begin{cases} 1 & \text{if } 10^\circ \leq \varepsilon \leq 50^\circ \\ \frac{1}{5}\varepsilon - 1 & \text{if } 5^\circ \leq \varepsilon \leq 10^\circ \\ 6 - \frac{1}{10}\varepsilon & \text{if } 50^\circ \leq \varepsilon \leq 60^\circ \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

The score retrieved from each face is then summed together and an average reward signal is obtained by dividing by the number of faces n_{faces}, as follows:

$$r_{\varepsilon_{avg}} = \frac{1}{n} \sum_{j=0}^{n} r_{\varepsilon,j} \tag{3.6}$$

- *Sun incidence score.* The Sun incidence angle η is the angle between the Sun direction relative to the target object and the normal to the face considered. The Sun incidence angle should be between 10° - 60°, to avoid shadows or excessive brightness. Values outside that interval may correspond to conditions that degrade the quality of the image. A unitary score is assigned for each of the faces in correct exposition among the n faces that are in the FoV of the spacecraft camera. Then, the overall score is given by the average expressed in Eq. 3.7.

$$r_{\eta_{avg}} = \frac{1}{n}\sum_{j=0}^{n} r_{\eta,j} \tag{3.7}$$

where $r_{\eta,j}$ is the Sun incidence score for a single face and is defined in Eq. 3.8.

$$r_\eta = \begin{cases} 1 & \text{if } 10^\circ \leq \eta \leq 60^\circ \\ \frac{1}{10}\eta & \text{if } 0^\circ \leq \eta \leq 10^\circ \\ 7 - \frac{1}{10}\eta & \text{if } 60^\circ \leq \eta \leq 70^\circ \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

- *Map level score*: to better reconstruct the target geometry and shape, a reward on the current level of the map is necessary. The overall map is fragmented into a number N_p of quality photos for each face constituting the mesh, where quality is to be intended with respect to the incidence angle ε between the camera and the face. At each time step, the map percentage can be computed by counting the number of quality pictures ($r_\varepsilon \neq 0$) available for each face N_q up to that moment and dividing this quantity by N_p times the number of mesh faces n_{faces}, as in Eq. 3.9. At each time step, the algorithm checks which faces of the mesh are in the visibility of the camera, and a picture of one of these faces is said to be of "good quality" if the reward signal r_ε associated with that single face is greater than zero.

$$M_{\%,k} = \frac{N_q}{N_p * n_{faces}} \tag{3.9}$$

$$r_m = \begin{cases} 1 & \text{if } M_{\%,k} > M_{\%,k-1} \\ 100 & \text{if } M_{\%,k} = 100 \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

In Eq. 3.10, note how the agent is rewarded for improving the map level and it is also given a big bonus for completing the map reconstruction. The condition in which the agent has successfully reconstructed the total target shape is intentionally not defined as a stopping condition. In this way, the agent is rewarded more if it rapidly completes the map. In general, an episode will end only if the spacecraft escapes the bounded region of space defined or if the defined simulation time runs out.

- *Time of flight score.* This score rewards the agent for obtaining the best map within a definite time window, identified by bounding the time of flight between T_{min} and T_{max}.

$$r_t = \begin{cases} 0 & \text{if } \Delta t < T_{min} \\ 1 & \text{if } T_{min} \leq \Delta t \leq T_{max} \end{cases} \tag{3.11}$$

  There is no score for $\Delta t > T_{max}$ because the simulation is stopped if reaches T_{max} and therefore the episode ends.

- *Thrust score.* This score considers the number of times that thrusters are fired, n_f. Two thresholds are assigned: the first is a medium-level threshold, l_{mid}, beyond which is still possible to fire the thrusters; the second threshold, l_{max}, defines the maximum number of firings, beyond which the thrusters can not be used again. The score is defined in Eq. 3.12.

$$r_f = \begin{cases} -10 & \text{if } l_{mid} \leq n_f < l_{max} \\ -100 & \text{if } n_f \geq l_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

The total reward received by the agent at each time step of the training simulation, therefore, is simply the sum of these components. For example, Eq. 3.13 considers only the distance, time and map level in the reward computation.

$$R = r_d + r_t + r_m \tag{3.13}$$

In the next chapters, different reward models will be defined and exploited for the agent training, in order to differentiate the objectives of the spacecraft to suit them for the scenario and the target object.

3.2 Visibility Model

The *visibility model* here explained is the mathematical model that, at each time step, generates the portion of the target object that the spacecraft (with its camera) can effectively observe. Considering the target shape modelled as a set of faces, *visibility* means not only locating which of them are facing towards the spacecraft camera but also, the ones illuminated by the Sun: thus, actually visible when the image is taken. Therefore, it is worth underlining the distinction between two of the terminologies that will be used in the following sections: *visible* and *observable*.

- **Observable**: is the face that is in the camera FoV but can not be actually photographed owing to a bad Sun exposure.
- **Visible**: is the face that is in the camera FoV and simultaneously exposed to the Sun.

As mentioned before, during part of the work, a strong assumption on the spacecraft's attitude has been made to simplify the computational load without departing from the ultimate goal of the study. Indeed, the spacecraft's camera is always assumed to point to the object CM. This leads to the neglect of the attitude dynamics of the spacecraft and therefore the attitude control, that, otherwise, would have been essential in order to get the image reconstruction of the target. This assumption is reputed valid until the goal is the autonomous control of the spacecraft trajectory around the object and not the control of the spacecraft to get the object in its field of view (this kind of assumption has been done also both in [15] and [14] without losing the aim of the study). If on the one hand, the use of attitude dynamic control would have generated a more general and realistic model in relation to the solution, on the other, the complexity that was going to be faced would have surely made the achievement of a good result much more hard for the learning agent. Therefore, at first, for all the analysis presented in Chapter 4, this additional complexity was not deemed necessary and advantageous. It will be observable, how the agent faces the additional complexity of attitude control in Chapter 5.

That said, the result of the so-called visibility model derives from the combination of the chaser-target relative dynamics previously described. Hence, once the chaser-target relative position and rotation are known, the following information can be retrieved:

- Direction of the camera towards the target object c.m.
- Direction of the Sun with respect to the target c.m.
- Normal vectors of each of the faces (composing the shape of the target)).

Therefore, considering the translational-rotational dynamics formulations in Section 2.1, these quantities are expressed in the LVLH-frame and thus consistent with each other. For this reason, the knowledge of the angles between the face normal and the direction of the Sun and the camera is enough to understand if that particular face is exposed to the Sun and the camera or not. Then a further check is needed to define if the angle is below the camera FoV: in that case, the face is considered *visible*. This leads to the three-step procedure described below:

1. Computation of the angle between the camera direction and the normal to the face. If it is below 90°, the face is facing towards the spacecraft.

2. Check on the FoV of the camera: if the angle computed in the step above is within the FoV then the face is observable to the camera.

3. Computation of the angle between the Sun's direction and the normal to the face. If it is below 90°, the face is exposed to the Sun and therefore visible by the camera.

In Fig. 3.1, an example of what is explained is presented. The plot shows an object composed of N number of faces: the ones coloured in red are in the FoV of the camera (represented by the light blue cone) and with good Sun illumination; instead, the faces in yellow are anyway in the FoV but with incorrect Sun exposure.

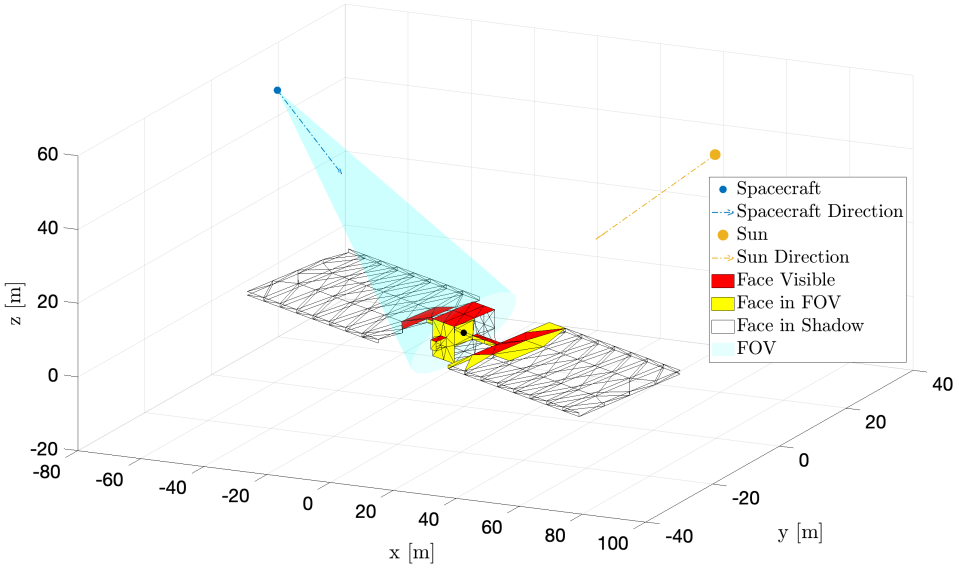


Figure 3.1: Visibility model practical example: differentiation between *observable* and *visible* faces [26].

It is interesting to analyse how this particular shape, here used to depict the target object, represents a great drawback of the assumption made on the fixed camera pointing. Indeed, in this case, it would be unlikely to have the solar panels in the camera's FoV because of their very elongated shape, except from a high distance between the two bodies. Therefore, this is an example of how the previous assumption reduces the possibility of target shape generalization.

Summing up, the *visibility* is affected by the quantities listed below:

- the camera FoV;
- the chaser-target relative distance;

- the incidence angle between the camera direction and the faces' normal;
- the incidence angle between the Sun direction and the faces' normal;

In particular, the last two are of crucial importance for the goodness of the image, and consequently, they are considered in the formulation of the reward model, discussed in Section 3.1.

Another strong assumption has been made: the absence of the main attractor, i.e. the Earth. Its presence would have posed two issues: the first is that, especially for low-altitude orbits, in which the orbital period is comparable to the mission time, the Sun would have been subjected to long and frequent shadowing periods. The second is that the Earth would have been present in several images as background, reducing the capabilities of distinguishing the object in the image (e.g. challenging issue for image processing exploited in Chapter 5). Therefore, although the Sun eclipses can strongly influence the success of a mission, as the simulated one, and the time in which it is accomplished, this aspect is not examined to avoid major implementation and complexity issues. A proper analysis of the target object Keplerian parameters has been performed in Section 7.2.1 in [26] in order to be consistent with this model simplification.

3.2.1 Uncooperative and Unknown Object Model

As highlighted in the chapter's introduction, three main scenarios will be analysed during the next chapters, characterized by three different target objects. The first one, depicted in Fig. 3.2, is just a rectangular-shaped object, needed to assess the baseline of the algorithm. In Eq. 3.14 the inertia matrix used for the relative simulation is defined.

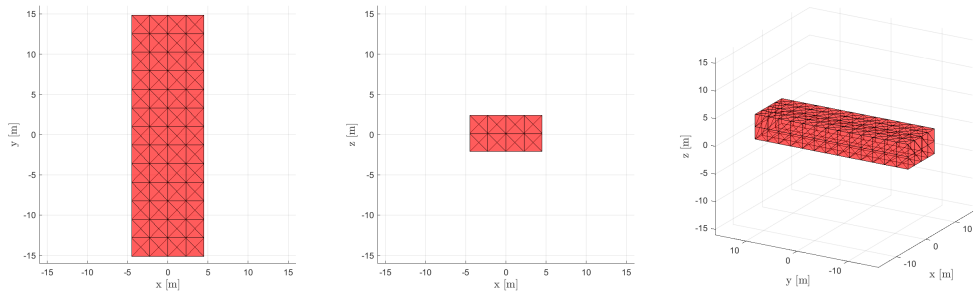


Figure 3.2: General-purposed rectangular target object mesh.

$$I = \begin{bmatrix} 76.69 & 0 & 0 \\ 0 & 8.44 & 0 \\ 0 & 0 & 81.75 \end{bmatrix} kgm^2 \quad M = 100kg \tag{3.14}$$

The second scenario treated sees the VESPA object fly-around, whose mesh and inertia matrix are shown in Fig. 3.3 and Eq. 3.15.

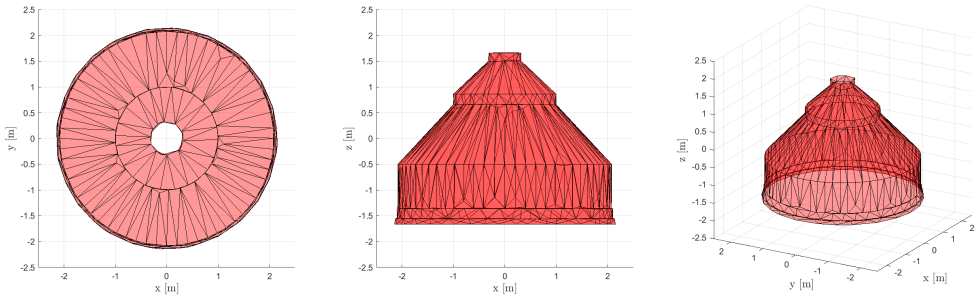


Figure 3.3: VESPA (Vega Secondary Payload Adapter) target object mesh.

$$I = \begin{bmatrix} 0.5208 & 0 & 0 \\ 0 & 0.5208 & 0 \\ 0 & 0 & 0.6667 \end{bmatrix} kgm^2 \quad M = 15kg \tag{3.15}$$

The problem scenario that will be analysed in Chapter 5 is characterized by a chaser that orbits around TANGO, defined by the mesh and inertia matrix in Fig. 3.4 and Eq. 3.16.

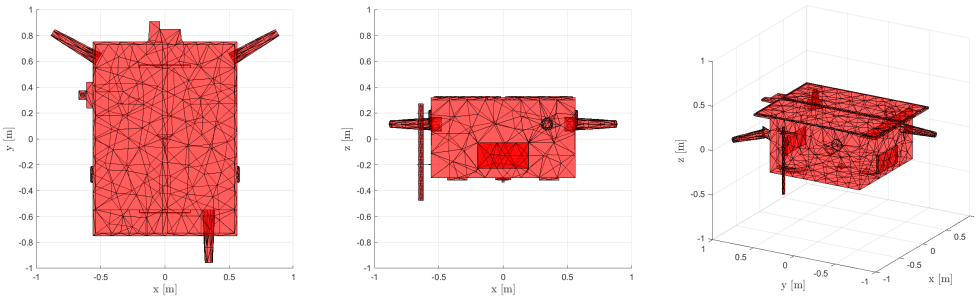


Figure 3.4: TANGO (from PRISMA mission) mesh.

$$I = \begin{bmatrix} 0.0618 & 0 & 0 \\ 0 & 0.0553 & 0 \\ 0 & 0 & 0.1003 \end{bmatrix} kgm^2 \quad M = 38.5kg \tag{3.16}$$

Several differences can be spotted between the three objects, not only in terms of inherent shape and inertia, but also concerning the mass, and the maximum and minimum length, which affect the agent behaviour related to trajectory control, and the number of faces, that instead affects the map reconstruction

model. These characteristics, denoted in Table 3.1, influence the training and testing simulations.

Object	Shape	Max Len.	# Nodes	# Faces
General-pur.	Rectangular	15 m	773	1549
VESPA	Symmetric, Conical	2 m	315	600
TANGO	Symmetric, Cubical	1 m	174	344

Table 3.1: Comparison between the different target objects mesh used in the thesis analysis.

3.3 Network Model

The choice of neural network architecture, such as Feedforward Neural Networks (FFNN), Recurrent Neural Networks (RNN), or others, can have different benefits and implications when applying PPO. Each of the architecture can have different advantages for PPO:

- **FFNN** are relatively simple and have a feed-forward structure with no internal memory, which can make them easier to train and implement. FFNNs can process multiple observations simultaneously, making them well-suited for environments where parallelism is important; and in environments where the Markov property holds (current state contains all relevant information), FFNNs can be an effective choice.
- **RNN** is designed to handle sequences of data and can capture temporal dependencies, making them a good choice for tasks where the agent's actions depend on a history of observations. RNNs are designed to have internal memory, which allows them to remember past observations and actions, making them suitable for partially observable environments. Moreover, they can capture long-term dependencies, which can be beneficial for certain tasks where actions have delayed consequences.
- **LSTM** (Long Short-Term Memory) and **GRU** (Gated Recurrent Unit) are specific types of RNNs that address some of the vanishing gradient problems associated with traditional RNNs. They provide a more robust way to handle long-term dependencies and may be a good choice for PPO in tasks with extended temporal dependencies.

The choice of architecture depends on the nature of the environment and the specific requirements of your reinforcement learning problem. FFNNs are often a good starting point due to their simplicity, but for tasks with sequential data or partially observable environments, RNNs, LSTM, or GRU networks may be more appropriate. Therefore, starting from these characteristics one of the first and main analyses carried out in this thesis was developed: the

comparison in terms of performance between two agents defined by a simple and classic multi-layer feed-forward neural network (FFNN), and a recurrent neural network (RNN).

In the following subsections, the two architectures will be presented in terms of depth, dimension and activation function. A deeper analysis regarding networks' hyper-parameters is carried out in Section 3.4.

3.3.1 Linear Architecture

Two main linear models will be exploited in this work: the first in Section 4.2, and the second in Section 4.3; both of them are the result of an extensive testing campaign in [26] and [66] to define their best depth and dimension for the discrete and continuous case. In both cases, the policy and value functions are equally defined.

The architecture for the discrete action space case is described in Table 3.2 and Table 3.3 and is composed of four linear hidden layers with different dimensions and activation function and takes inspiration from [19]: where the first and last hidden layers are ten times greater than the state input (defined in Section 3.1.1) and action output (defined in Section 3.1.2) layers respectively. The second and third layers are identical except for the activation function with a dimension equal to the square root of the product between the element number of the first and fourth. In order to improve the convergence and avoid saturation problem the tanh-layers are initialized as semi-orthogonal matrices, as suggested by [68].

		Policy Network	
Layer	Type	Elements	Activation
Input	Linear	12	tanh
Hidden Layer - 1 (h1)	Linear	10*12 = 120	tanh
Hidden Layer - 2 (h2)	Linear	$\sqrt{n_{h1} * n_{h4}} = 92$	tanh
Hidden Layer - 3 (h3)	Linear	$\sqrt{n_{h1} * n_{h4}} = 92$	Leaky-ReLU
Hidden Layer - 4 (h4)	Linear	10*7 = 70	Leaky-ReLU
Output	Linear	7	softmax

Table 3.2: Policy (Actor) Network Architecture: linear case for discrete action space.

In Table 3.4 and Table 3.5, the definition of the linear architecture for the continuous action space case is defined. In this case, only two hidden layers are necessary for the optimization: both of them have the same length and activation function.

		Value Network	
Layer	Type	Elements	Activation
Input	Linear	12	-
Hidden Layer - 1 (h1)	Linear	10*12 = 120	tanh
Hidden Layer - 2 (h2)	Linear	$\sqrt{n_{h1} * n_{h4}} = 92$	tanh
Hidden Layer - 3 (h3)	Linear	$\sqrt{n_{h1} * n_{h4}} = 92$	Leaky-ReLU
Hidden Layer - 4 (h4)	Linear	10*7 = 70	Leaky-ReLU
Output	Linear	7	linear

Table 3.3: Value (Critic) Network Architecture: linear case for discrete action space.

		Policy Network	
Layer	Type	Elements	Activation
Input	Linear	12	-
Hidden Layer - 1 (h1)	Linear	256	ReLU
Hidden Layer - 2 (h2)	Linear	256	ReLU
Output	Linear	1	tanh

Table 3.4: Policy (Actor) Network Architecture: linear case for continuous action space.

		Value Network	
Layer	Type	Elements	Activation
Input	Linear	12	-
Hidden Layer - 1 (h1)	Linear	256	ReLU
Hidden Layer - 2 (h2)	Linear	256	ReLU
Output	Linear	1	-

Table 3.5: Value (Critic) Network Architecture: linear case for continuous action space.

In this PPO implementation, both the policy and the value functions (actor and critic networks) are learned concurrently. In Table 3.2, the action space is discrete, thus implying the use of a softmax activation function to select the action at each time step, differently for the continuous case in Table 3.4, in which a sigmoid function is utilized. The output of the softmax activation function is a multi-categorical distribution, among which the policy samples the action to take during the optimization process. Differently, the output of the continuous case is directly used as the action state. The main hyper-parameters assigned to the optimization will be discussed in Section 3.4.

3.3.2 Recurrent Model

The idea behind the formulation of the PPO with recurrent neural networks is related to the potential benefits that such an architecture can have in terms of training and performance. Since recurrent networks have the capability to store past states' information, they may strongly benefit the agent-safe trajectories planning to faster achieve the mission goals. In addition, training an RNN may be beneficial to refine the agent's environmental conditions sensitivity, increasing its robustness, regardless of the specific operational environment. Therefore, we considered it important to compare both architectures to understand if an improvement of the stability and sensitivity can be possible with respect to the particular conditions in which the problem is solved.

The architecture of both policy and value functions is equally defined coupling two LSTM recurrent layers and two drop-out linear layers [48]. The two models are shown in Table 3.6 and Table 3.7. The only differences between the two are the linear layers' activation functions which tanh for policy and ReLU for value network.

		Policy Network	
Layer	Type	Elements	Activation
Input	Linear	12	-
Hidden Layer - 1 (h1)	LSTM	24	-
Hidden Layer - 2 (h2)	LSTM	24	-
Hidden Layer - 3 (h3)	Linear	100/64	tanh
Hidden Layer - 4 (h4)	Linear	100/32	tanh
Output	Linear	7	softmax

Table 3.6: Policy (Actor) Network Architecture: Recurrent Case

		Value Network	
Layer	Type	Elements	Activation
Input	Linear	12	-
Hidden Layer - 1 (h1)	LSTM	24	-
Hidden Layer - 2 (h2)	LSTM	24	-
Hidden Layer - 3 (h3)	Linear	100/64	ReLU
Hidden Layer - 4 (h4)	Linear	100/32	ReLU
Output	Linear	7	linear

Table 3.7: Value (Critic) Network Architecture: Recurrent Case

3.4 Hyperparameters Analysis

In deep reinforcement learning, hyper-parameters are the whole settings which are not learned during the training process but may significantly impact the model's learning and performance. Tuning these hyper-parameters effectively can greatly influence the stability, convergence speed, and overall performance of algorithms and training simulations. Therefore, before entering into the details of the autonomous guidance agent analysis, it is worth to briefly recap and summarize all the hyper-parameters which must be inevitably selected to design and run the simulations. Although it may seem trivial, this is a critical point to correctly shape the training step. In general, or at least for the following analysis, the set of parameters to be defined comprehend the following variables:

- Number, size, and type of hidden layers; type of activation function between networks' layers. All these parameters have been already defined for each of the different architectures presented in the previous Section 3.3.
- Optimizer method (Adam, RMSprop, SDG, etc.) and DRL algorithm optimization-related parameters such as learning rate, which controls the step size during the learning process; reward discount factor γ, the factor that multiplies the reward at each time step determining the importance of future rewards compared to immediate rewards; the terminal reward discount factor λ, which defines the factor multiplying the overall sum of rewards at the end of a single episode; the clipping parameter ϵ (strictly related to the PPO algorithm) that limits the magnitude of gradients during training; entropy coefficient c_2 which defines the balance between exploration and exploitation at each optimization step. Also, the number of epochs is another important parameter which sets the number of network updates at each step.
- Memory size, simulation size and length: the first defines the capacity of the memory buffer to store experiences for replay, which is afterwards divided into batches of fixed size used in the training iterations. Instead, the simulation length can vary depending on the convergence speed or problem complexity.

In addition to the ones just listed, several other parameters may be needed considering different optimization algorithms, e.g. for DQN the target network update frequency is required. Also, the kind of network initialization, or regulation techniques may represent another hyper-parameter to tune, if used.

In conclusion, in Table 3.8, all the hyper-parameters, which will be exploited in the following chapters, are collected and their range of values is defined. These parameters will be specified in detail for each type of training analysis.

Hyper-parameter		Range
Optimizer	-	Adam
Learning rate	l	1e-4-1e-6
Discount factor	γ	0.95-0.99
Terminal discount factor	λ	0.95
Clipping parameter	ϵ	0.1/0.2
Entropy coefficient	c_2	0.01-0.1
Epochs	-	8-10
Memory size (*in episodes*)	-	10
Batch size (*in steps*)	-	32
Simulation length/iterations	-	1000-3000

Table 3.8: Hyper-parameters ranges.

CHAPTER 4

Autonomous Guidance & Control in Relative Dynamics Scenario

Design of a mildly unstable guidance and control system: application to very short space missions.

— Andrea Capannolo, *Titoli di paper che non verranno mai scritti*

Starting from the environment model defined in Section 3.1, and the network model in Section 3.3, in this chapter the DRL-based spacecraft guidance in relative dynamics scenario is studied and analysed. At first, in Section 4.1 an overview of the python-based tool is presented; in Section 4.2 the baseline case is analysed especially in terms of training flexibility and initial conditions robustness. Afterwards, in Section 4.3 and Section 4.4 the two main assumptions on *discrete* action space and *perfect* input state are relaxed and deeply analysed. In the end, in Section 4.6, a benchmark analysis is carried out starting from the results obtained in [26] and [66].

4.1 Python-based Tool

In Section 2.4 an overview of the most important existing software tools to solve DRL problem has been introduced. As denoted, most of them are developed within the Python framework. Also, this is the case, in which a PyTorch-based algorithm has been developed. The tool architecture is designed similarly to the `gym` environment by OpenAI[1], even if the library was not used to have increased flexibility in the functions' definitions proper of the scenario studied in the thesis. Nevertheless the architecture, as in the `gym` library, is built with a high level of versatility, in order to be easily adaptable to different environments (i.e. target object, initial conditions, chaser setting and objectives) or algorithms (i.e. types of NN, DRL algorithms, loss functions, hyper-parameters).

In particular, the architecture is structured as follows:

```
Environments/
input_netwroks/
input_templates/
env/
Learning Algorithms/
ppo_algorithm.py
a2c_algorithm.py
Neural Networks/
Mesh/
Output/
Utils/
main.py
```

Listing 4.1: Python-based tool main structure

where each environment is defined by the following list of functions:

```
env1/ # (e.g. map_uuso)
dyn.py
env.py
inc.py
loader.py
main_test.py
main_train.py
post.py
rew.py
utils.py
env2/ # (...)
```

Listing 4.2: Python-based tool environment structure

The environment folder, then, is composed by:

- `dyn.py`: in which the equation of the dynamics is defined.

[1] https://www.gymlibrary.dev/index.html

- `inc.py`: that defines the initial condition, if fixed or random.
- `loader.py`: is used to load all the simulation setting information derived by a `.yaml` file.
- `rew.py`: in which all the possible reward functions are defined.
- `env.py`: puts together all the previous functions and classes to create the environment.
- `main_train.py` and `main_train.py`: connect the environment function and the algorithm to train and test the agent.
- `utils.py` and `post.py`: define utility functions and post-processing functions respectively.

In Fig. 4.1 and Fig. 4.2 the schematics of the constructed codes for training and testing an environment are depicted. The main difference between the two is obviously in the *algorithm* part: the training exploits both policy and value networks to retrieve the action and value parameters respectively in order to increase the memory class which will be used in the proper training step. The testing architecture uses only the policy network to sample the action and close the loop with the environment.

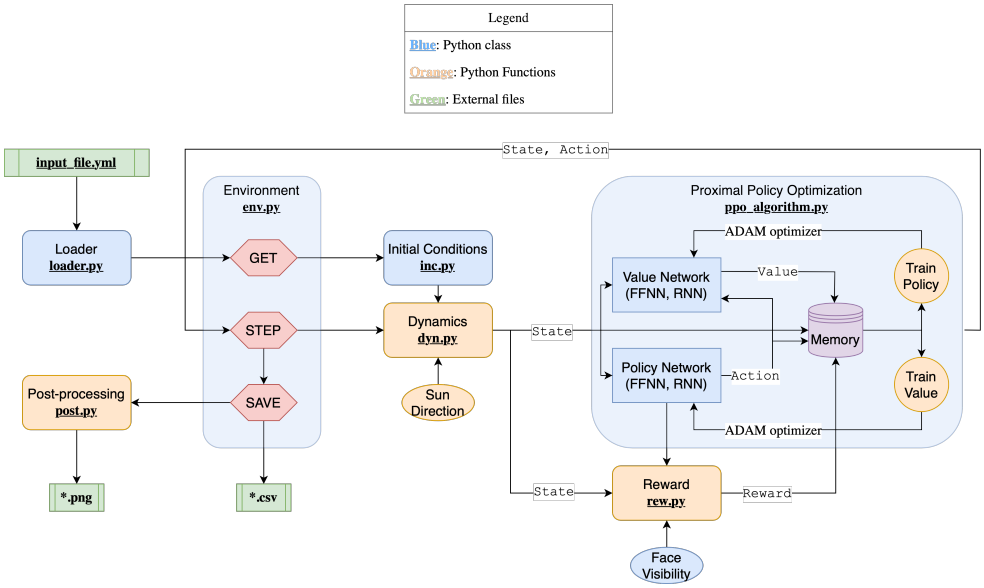


Figure 4.1: Schematic of the python-based tool for training simulation `main_train.py`.

A lot of parameters, features and variables can be modified within the code framework such as the reward function, the network model, the initial condi-

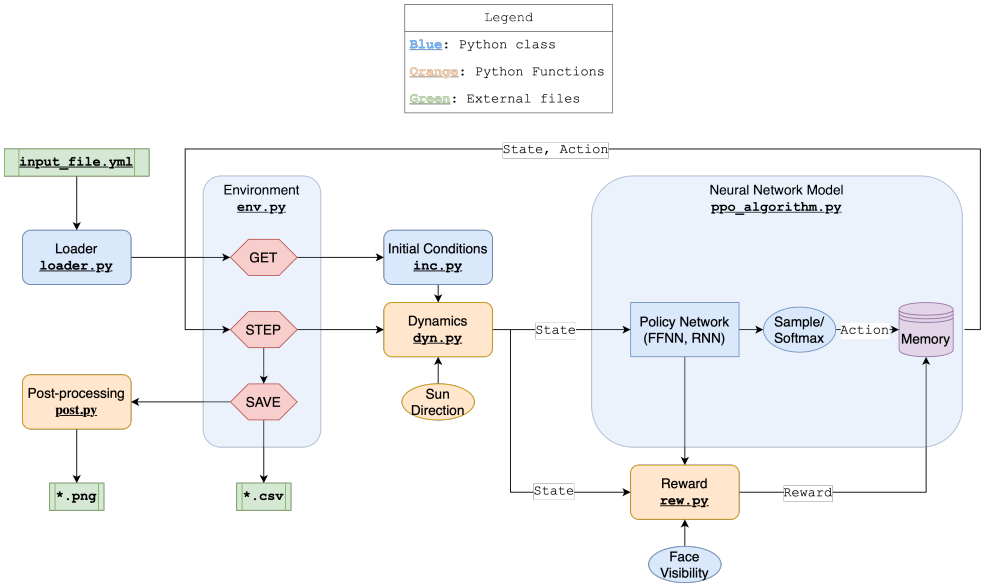


Figure 4.2: Schematic of the python-based tool for testing simulation `main_test.py`.

tions, the environment settings, the algorithm optimizer (i.e. Adam) and so on. The `input_file.yml` keeps track of everything.

4.2 Baseline Case: training & testing

In this section, the baseline case, needed to create and define the foundations of the work, is developed and explained. It has the goal of the map reconstruction of an uncooperative and unknown target object. As a baseline, the general-purpose rectangular object is selected for the target, as defined in Fig. 3.2. Some ground basis characteristics proper of the environment formulation will be defined in the next subsection such as the camera field-of-view, the integration step, the trajectory distance range and so on. It is worth to underline that all these features may change from case study to case study, that is why it has been decided to highlight them also in a green box. The most important purposes of this baseline case are three:

- **Robustness study** especially in terms of initial conditions. This analysis is split between the training phase, in which different levels of initial conditions have been required, and also the testing phase where different environment characteristics have been modified in order to understand the limits of the adopted architectures.

- **Mission flexibility** concerning the analysis related to the reward shaping and how the training architecture is flexible to changes in the reward, i.e. different scores selected.
- **Network comparison** of the two proposed architectures: feed-forward fully-connected and recurrent ones. This analysis aims to figure out how the training phase can benefit from the main features of the two models.

In order to maintain the orientation between the several analyses carried out and the different models used the two phases have been divided into two cases as outlined in the next subsection. Finally, before entering into the details of the training and testing analysis it is worthy to recap all the networks' models and algorithm's hyper-parameters used for the simulations. The policy and value networks are shaped as the ones presented in Section 3.3 even if smaller as defined in Table 4.1 and Table 4.2, for the linear and recurrent architectures respectively. HL stands for *hidden layer*, while DOHL means *drop-out hidden layer*.

	Policy Network		Value Network	
Layer	Elements	Activation	Elements	Activation
Input	12	-	12	-
HL1	120	tanh	120	tanh
HL2	92	tanh	92	tanh
HL3	70	Leaky-ReLU	70	Leaky-ReLU
Output	7	softmax	7	-

Table 4.1: Policy and value feed-forward fully-connected network model for the baseline training & testing case.

	Policy Network		Value Network	
Layer	Elements	Activation	Elements	Activation
Input	12	-	12	-
LSTM Layer	24	-	24	-
DOHL1	64	ReLU	64	ReLU
DOHL2	32	ReLU	32	ReLU
Output	7	softmax	7	-

Table 4.2: Policy and value recurrent network model for the baseline training & testing case.

The set of hyper-parameters defined in Table 4.3 comes from numerous preliminary analyses aimed at assessing the best combination of them and tuning the algorithm in order to assure a good convergence of the methodology. They have been differentiating between the algorithm's parameters, policy and value network ones.

PPO Hyperparameters	
Reward Discount Factor γ	0.99
Terminal Reward Discount Factor λ	0.95
Clipping Factor ϵ	0.2
Entropy Factor s_2	0
Optimizer	ADAM
Optimization Step	each 10 episodes
Optimization Batch	32
Epochs	5
Simulation length	$>$ 15000 episodes (training)
Policy Network	
Initialization (for *tanh* layers)	Orthogonal
Initialization (for *relu* layers)	He-uniform
Learning Rate	5e-5 (FFNN) 6e-5 (RNN)
Final Activation Function	Sigmoid
Output Distribution	Categorical
Output Action	Sampling (training) argmax (testing)
Value Network	
Initialization (for *tanh* layers)	Orthogonal
Initialization (for *relu* layers)	He-uniform
Learning Rate	5e-5 (FFNN) 6e-5 (RNN)
Final Activation Function	-

Table 4.3: Set of PPO algorithm's hyper-parameters used for training and testing of the baseline case study.

The overall training or testing simulation is composed of a pre-defined number of episodes, which can terminate in three possible terminal conditions, shaped in such a way that the algorithm's convergence can benefit from them:

- complete acquisition of the target object map, which depends on the number of mesh's faces and the selected accuracy level;
- chaser trajectory escaping the region defined by the minimum and maximum distance from the target;
- chaser agent exceeding the max time window.

All of these termination conditions rely on the definition of some environment parameters or features that may vary from one simulation to another, and that will be outlined each time a new case study is analysed.

In Table 4.4 the Keplerian elements of the target object are defined. As noted in Section 2.1 they do not have a serious influence on the trajectory for a time window whose length is one day at maximum. Therefore, concerning the Sun-Earth-Target inertial framework only two variables are allowed to change: the target true anomaly θ and the Sun phase with respect to the Earth θ_{Earth}.

a	**e**	**i**	**Ω**	**ω**	θ	θ_{Earth}
9000 km	0.01	2°	10°	10°	0°-360°	0°-360°

Table 4.4: Target object average initial conditions defined as Keplerian elements.

4.2.1 Training Analysis

In order to bound the problem some characteristics have been maintained constant during the overall training procedure. In particular, the camera field of view (FoV) is fixed at 10° (that can be considered as a common FoV for space optical cameras); the integration time is fixed at 30s (in a time window of a day: 86400s) and the accuracy level for the map at 25 correct (in terms of Sun illumination and camera orientation) photos per face. The chaser trajectory region is defined by a minimum and maximum distance from the target set as 1.25 and 40 of the object's maximum length (around 15m) respectively. The agent exploits a discrete action space equal to the one introduced in Section 3.1.2 with a thrust level set at 0.001 m/s^2. Instead, considering the chaser and target initial conditions, two levels of randomness have been considered:

- The first case considers a random Sun initial phase, a random target initial conditions, in terms of orbital true anomaly and rotational dynamics (angular position and velocity). Here, the chaser-target initial relative position is considered fixed. For simplicity, this case is named ***Random Case A***.

Random Case A
Chaser-Target: rel. position/velocity fixed, attitude random
Orbit features: Sun phase random, true anomaly θ random
Camera FoV: 10°
Integration step: 30 s for max 86400 s
Trajectory range: 18.75-600 m
Thrust level: 0.001 m/s^2
Map level: 25 photos/face on **Rectangular** shaped object

- The second case presents again a random Sun initial phase, a random target initial conditions, in terms of orbital true anomaly and rotational dynamics (angular position and velocity) and the chaser-target initial relative position. Their relative position is however constrained to have both the x, y and z coordinates positive. This assumption comes from two reasons in particular: firstly the will to contain the complexity of the problem in order not to saturate the neural network learning capabilities and also simulate a possible real scenario, in which the initial condition is constrained in a specific space without knowing a priori the correct engagement position. Similarly to before this case is named ***Random Case B***.

Random Case B
Chaser-Target: rel. velocity fixed, position (in +x, +y, +x space) and attitude random
Orbit features: Sun phase random, true anomaly θc random
Camera FoV: 10°
Integration step: 30 s for max 86400 s
Trajectory range: 18.75-600 m
Thrust level: 0.001 m/s^2
Map level: 25 photos/face on **Rectangular** shaped object

Aiming to better understand the DRL agent behaviour, in both the two cases, the neural networks were trained with two different reward models. Considering the different reward scores defined in Section 3.1.3, at first the agent is trained with only map, distance and time objectives, without caring about the thrust level, as outlined in the following expression:

$$R_{k,1} = r_m + r_d + r_t \tag{4.1}$$

where r_m, r_d and r_t are the scores defined in Eq. 3.10, Eq. 3.4 and Eq. 3.11 respectively. Afterwards, the complexity of the mission goal has been increased introducing the thrust level score r_f as defined in Eq. 3.12. In this second case the total reward is given by:

$$R_{k,2} = r_m + r_d + r_t + r_f \tag{4.2}$$

4.2.1.1 Random Case A

In Fig. 4.3, the results obtained with random target initial condition, random Sun phase and fixed relative position are shown. Here the $R_{k,1}$ reward model is exploited to make the agent learn to reach the best possible mapping quality. In

the plot, the average map level's trends of the linear (depicted as MLP-3) and recurrent (named as LSTM) policy architectures are compared in a simulation of 15000 episodes length.

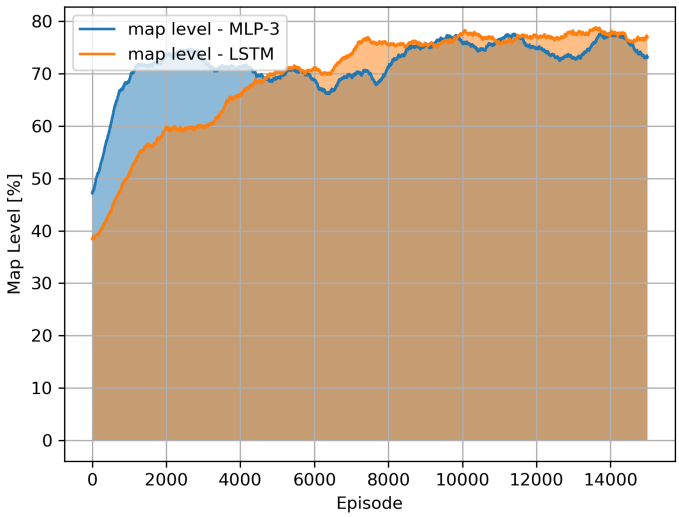


Figure 4.3: Random Case A. Map Level trend for reward model $R_{k,1}$ with position, map and time scores. Comparison between the two Policy Architectures: linear/MLP-3 and recurrent/LSTM.

Analysing the results some remarks can be derived:

- The linear policy, MLP-3, seems to learn and converge faster than the recurrent policy. Nevertheless, the MLP-3 curve presents more oscillations and an overall lower stability with respect to the LSTM curve policy behaviour.
- Concerning the final result of the simulation, the recurrent policy converges to a slightly higher map level in the same training length as the linear policy. On average the map level reached by the two policies is around 70%-80%.
- The fact that the learning curve of the LSTM policy grows gradually shows that having part of the network composed of recurrent layers makes the learning process slower, as expected, but safer and more stable. Indeed, the potential robustness of a recurrent network was one of the main reasons that drove this kind of analysis.

In Fig. 4.4, the same comparative analysis shown before was performed. In this case, the reward model, $R_{k,2}$, includes also the thrust level minimization

objective. As before, the same features found before are present again in this kind of simulation. MLP-3 is faster in learning but unstable in keeping on growing with respect to LSTM. The overall result is around 60%-70%. Not surprisingly, the percentage is slightly less than the one obtained with the $R_{k,1}$ reward model; indeed, the fact that the reward adds a new task to the agent determines a shift in the learning process priorities and therefore an automatic reduction of the map level if considering the same amount of episodes.

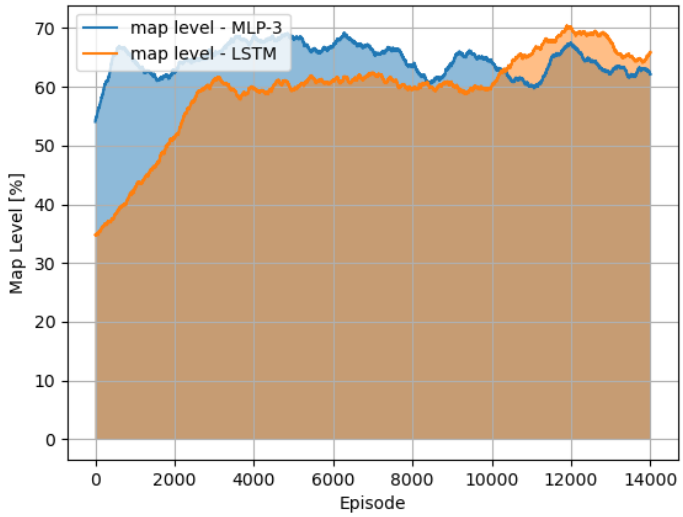
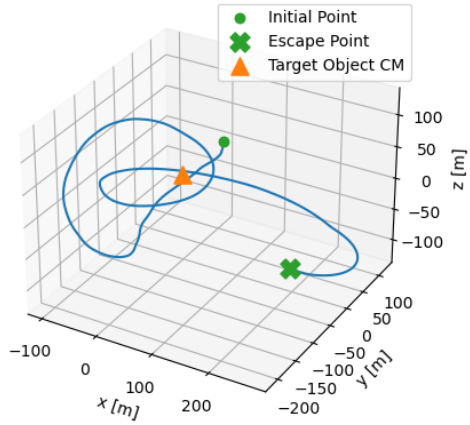


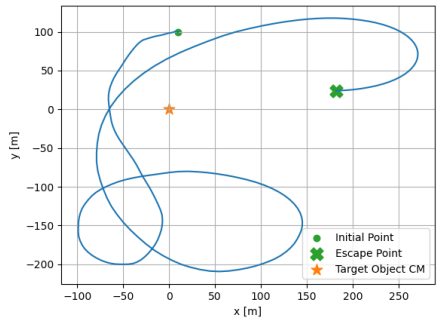
Figure 4.4: Random Case A. Map Level trend for reward model $R_{k,2}$ with position, map, time and thrust scores. Comparison between the two Policy Architectures: linear/MLP-3 and recurrent/LSTM.

It is worth noticing how the current method outperforms the one adopted in previous works [26], where the A2C algorithm was exploited. Indeed here with both the linear and the recurrent policy architecture the agent is capable of achieving an average map level of 70%-80%; with the A2C algorithm, with equal characteristics in terms of random initial conditions, the results were quite lower, around 40%. In Fig. 4.5, a resulting trajectory is shown. The agent, characterized by a recurrent policy and trained with $R_{k,1}$ reward model, starting from the fixed initial relative position of $r_0 = [10\text{m}, 100\text{m}, 10\text{m}]$ reaches a 100% map level in about 3.67h around the target object before escaping.

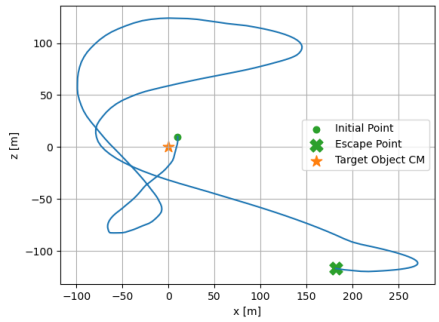
In Fig. 4.6, a sensitivity analysis on the thrust level has been performed. The analysis exploited the already trained linear, MLP-3, networks for the two different reward models. The aim of the analysis is to understand if the additional score, related to the thrust level, actually helps the agent to reduce the number of firings. In particular, it can be noticed that for the same type



(a) 3D Trajectory



(b) Trajectory XY-projection

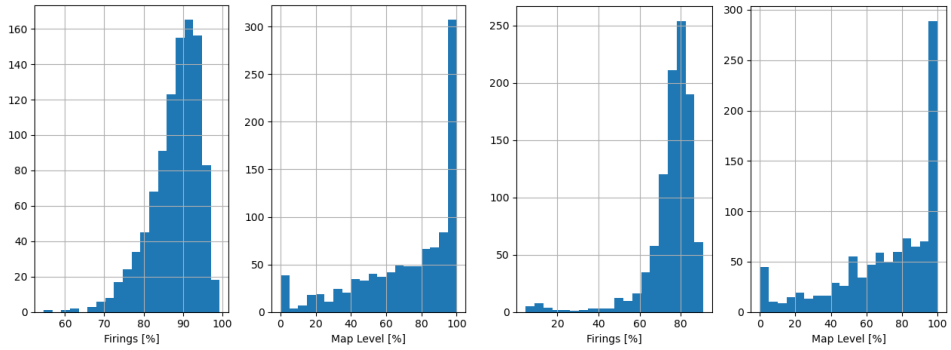


(c) Trajectory XZ-projection

Figure 4.5: Random Case A. Trajectory obtained with recurrent policy.

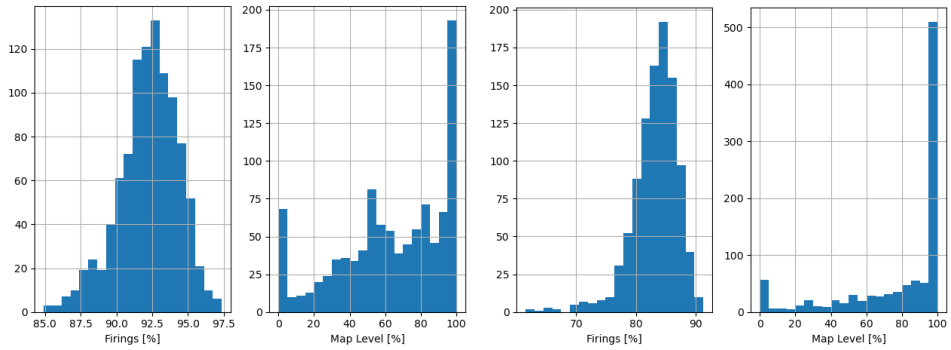
of policy architecture, the average thrust level is lower for the $R_{k,2}$ reward model. This result is expected because the reward model considers also the score relative to the containment of the number of firings. The value shown in the plot, that is the percentage of the firing is computed as the percentage of the firings selected by the agent with respect to all the possible firings that the agent could have chosen. As observable, in the $R_{k,1}$ model the average number of firings is around 90%, while in the $R_{k,2}$ model the average number is around 80%. Despite the results do not show a strong reduction in firing percentage, they demonstrate how the use of a specific thrust score is effective in accomplishing the task requested. The same result is obtained also considering the recurrent policy as depicted in Fig. 4.7.

In Figure 4.8, as an example, a trajectory obtained with the $R_{k,2}$ reward model by the linear policy is shown. In this plot, it is possible to observe the control action profile during the episode. In particular, the agent obtained this result



(a) Reward model $R_{k,1}$ **(b)** Reward model $R_{k,2}$

Figure 4.6: Random Case A. Thrust level comparison between the two reward models $R_{k,1}$ and $R_{k,2}$, with policy architectures: linear/MLP-3.



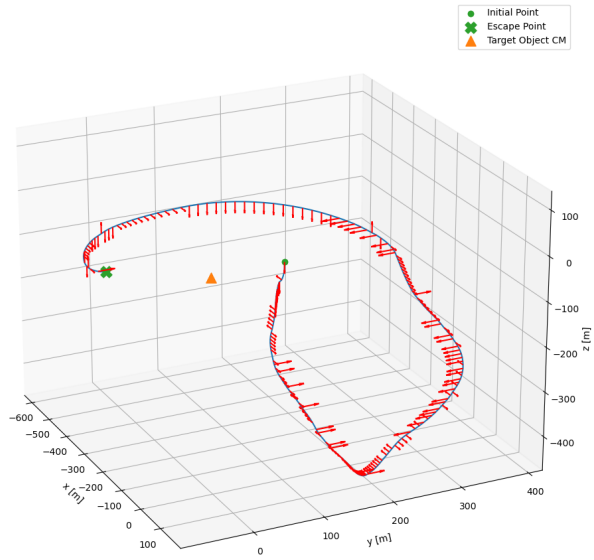
(a) Reward model $R_{k,1}$ **(b)** Reward model $R_{k,2}$

Figure 4.7: Random Case A. Thrust level comparison between the two reward models $R_{k,1}$ and $R_{k,2}$, with policy architectures: recurrent/LSTM.

reaching a map level of 94%, along a trajectory which lasted 2.1h with a 75.4% of firings.

4.2.1.2 Random Case B

In Fig. 4.9, the results obtained with random target initial condition, random Sun phase and random initial relative position are shown. Here the $R_{k,1}$ reward model is exploited to make the agent learn to reach the best mapping quality possible. In the plot, the average trends of the Linear and Recurrent policy architectures are compared for 15000 episodes simulation. The level reached by the two policies is comparable, around 55%. Also here, as in the first two analyses shown, the characteristics of the different architectures hold. As



(a) 3D Trajectory

Figure 4.8: Random Case A. Trajectory obtained with linear policy.

expected, the average map level is lower than the one achieved in the *Random Case A*; this outcome was expected because the state space is quite bigger now, due to the randomness in the initial relative position. As before, it is worth to underline that the results obtained with a PPO learning process greatly overcome the ones obtained by A2C in [26].

In Fig. 4.10, the same comparative analysis is performed. In this case, the reward model, $R_{k,2}$, includes also the thrust level minimization objective. In this analysis, differently from the previous ones, the linear policy performs better than the recurrent policy with an equal learning process length. Some features are equal, like the different growth rates, especially for the recurrent policy. The reason behind the fact that the linear policy reaches a higher level after 15000 episodes is related to the combination of the reward model, $R_{k,2}$ and the randomness level. Indeed, here, the state space is bigger and also the objectives are more complex and wider; this determines the growth of the recurrent policy to be slower than in the other analysis. However, for both the architectures the level reached is around 50%; in terms of map level, this result is not as good as the ones obtained with $R_{k,1}$. Nevertheless, considering the fact that for the current reward model, the state space is greatly wider, the result is high enough to be considered good.

In Fig. 4.11, the sensitivity analysis of the thrust level is shown. In this case, the analysis exploited the already trained recurrent, LSTM, network for the

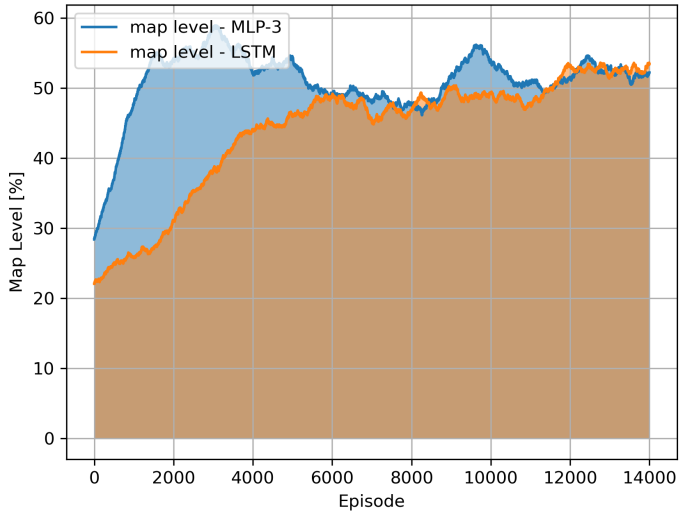


Figure 4.9: Random Case B. Map Level trend for reward model $R_{k,1}$ with position, map and time scores. Comparison between the two Policy Architectures: linear/MLP-3 and recurrent/LSTM.

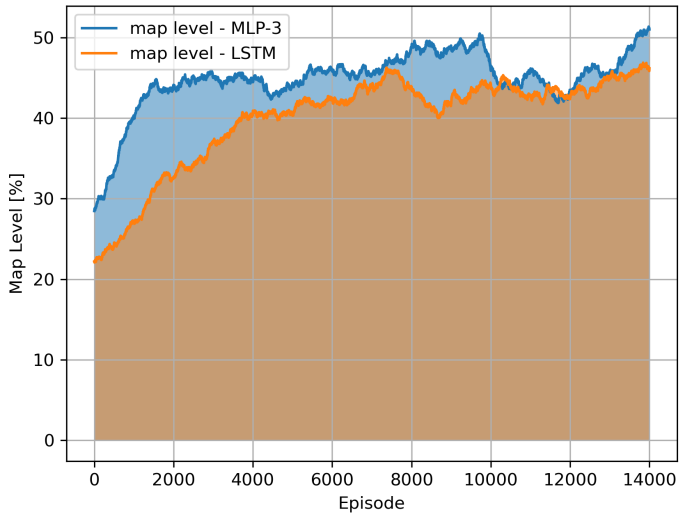
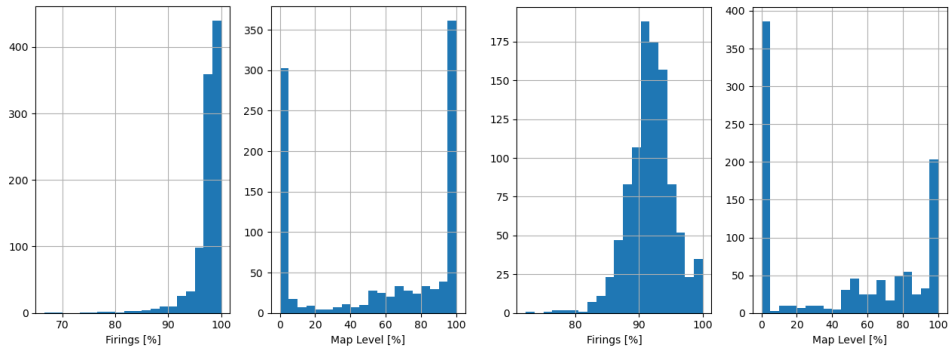


Figure 4.10: Random Case B. Map Level trend for reward model $R_{k,2}$ with position, map, time and thrust scores. Comparison between the two Policy Architectures: linear/MLP-3 and recurrent/LSTM.

two different reward models. Also here, it can be noticed that for the same type of policy architecture, the average thrust level is lower for the $R_{k,2}$ reward model.



(a) Reward model $R_{k,1}$ **(b)** Reward model $R_{k,2}$

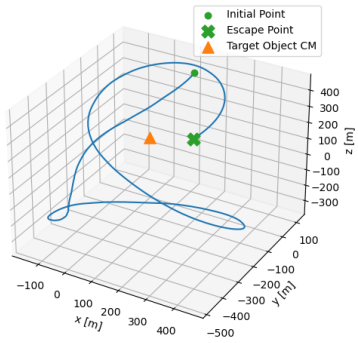
Figure 4.11: Random Case B. Thrust level comparison between the two reward models $R_{k,1}$ and $R_{k,2}$, with policy architectures: recurrent/LSTM.

As observable, for the $R_{k,1}$ model the average number of firings is around 95%-100%, while for the $R_{k,2}$ model the average number is around 90%. The values are higher than the ones in *Random Case A* always because the state space is wider and therefore the agent needs stronger control to follow the objectives from different initial relative positions.

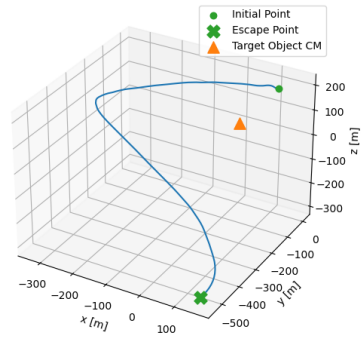
In Fig. 4.12 some trajectories obtained with the recurrent policy are shown. As observable, the initial relative position is different for each of them.

4.3 Action Space Analysis

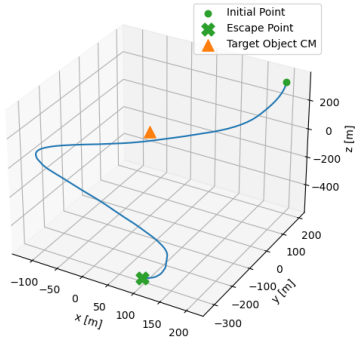
In this section, the transition from *discrete* action space to *continuous* is discussed [66, 69]. The major differences between the two have already been highlighted in Section 3.1.2, and therefore here, a new PPO agent, that works with a continuous action space is developed and analysed in two slightly different scenarios. This type of control workspace is much more realistic with respect to a discrete one, but at the same time, it is much more computationally expensive, due to the high dimensionality that the policy needs to analyse. Starting from the tests performed in the previous section on the baseline case, a feed-forward fully-connected model has been preferred over a recurrent model to give priority to fast learning at the expense of learning stability. After several preliminary simulations, based on the assumptions stated in Section 3.4, the policy and value networks are the ones already defined in Section 3.3. All the hyper-parameters used for the training are listed in Table 4.5.



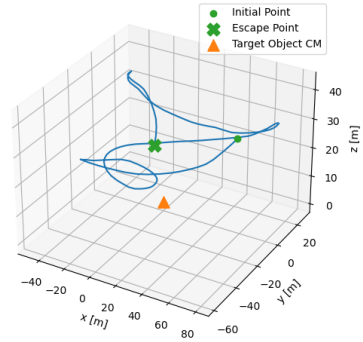
(a) 3D Trajectory - Map: 76.1% - Time: 2.21h



(b) 3D Trajectory - Map: 84.3% - Time: 1.27h



(c) 3D Trajectory - Map: 50.7% - Time: 1.06h



(d) 3D Trajectory - Map: 39.0% - Time: 1.08h

Figure 4.12: Random Case B. Trajectories obtained with recurrent policy.

For this analysis the reward function considers a vision-based system employing multi-spectral cameras, so both visible and thermal infrared. Therefore, all the results are referred to the $R_{Vis-Tir}$ reward expression, computed exploiting the distance, incidence and map scores as defined in Section 3.1.3. Notably this objective function is simpler with respect to the one considered in the baseline case, but this selection is justified by the intrinsic complexity of having a continuous action space, which would require much longer training.

$$R_{Vis-Tir} = r_d + r_{\varepsilon_{avg}} + r_m \tag{4.3}$$

Two scenarios have been studied:

- At first, the same environment treated in the baseline case is simulated with the new agent. The orbital target and Earth conditions are the same as defined in Table 4.4.

PPO Hyperparameters Continuous Action Space	
Reward Discount Factor γ	0.99
Terminal Reward Discount Factor λ	0.95
Clipping Factor ϵ	0.2
Entropy Factor s_2	0.001
Optimizer	ADAM
Optimization Step	each 5120 time-steps
Optimization Batch	512
Epochs	4
Simulation length	$>$ 20000 episodes (training)
Policy Network	
Initialization	Orthogonal
Learning Rate	1e-5
Final Activation Function	Tanh
Output Distribution	Normal
Output Action	Sampling (training)
Value Network	
Initialization	Orthogonal
Learning Rate	5e-5
Final Activation Function	-

Table 4.5: Set of PPO algorithm's hyperparameters used for training the continuous action space case study.

- Afterwards, a different target is considered, replacing the rectangular parallelepiped with a triangular mesh of VESPA (Vega Secondary Payload Adapter), as depicted in Fig. 3.3, which is a space object orbiting the Earth as debris with great interest by space agencies, defined as target for future missions [70, 64]. In Table 4.6 the VESPA orbital Keplerian elements are outlined.

a	e	i
6878 km	0.0096	98°

Table 4.6: VESPA orbital initial conditions.

It will be noticed that for this new agent, the training will be much longer than for the discrete action space agent. Once again this is justified by the much higher dimensionality of the problem, and by the generality of the initial

conditions, which are generated randomly for both the relative position and target attitude, as defined in Table 4.7.

Variable	Type		Range
Relative Position	Random	r	$2D_{min} < r < 0.5D_{max}$
		α	$0° < \alpha < 360°$
		β	$-90° < \beta < 90°$
Relative velocity	Fixed	v	$0\,m/s$
Angular position	Fixed	θ_i	$0°$
Angular velocity	Random	$\dot{\theta}_i$	-0.001 rad/s $< \dot{\theta}_i <$ 0.001 rad/s

Table 4.7: Relative chaser-target dynamics initial conditions in both scenarios: general-purposed rectangle object and VESPA.

Where D_{min} and D_{max} are the two boundaries, defined starting from the max dimensions of the two objects. Since a VIS-TIR camera is employed, no need for Sun information must be taken into account, relying on the double spectral information both for the navigation and the map reconstruction. For this reason, the single episode time window will be shorter with respect to the one defined for the baseline case.

General-purposed case. The first scenario is defined by the environment in the following box.

Continuos Space Agent - Rectangular Object
Chaser-Target: rel. velocity fixed, position and attitude random
Camera FoV: 10°
Integration step: 1 s for max 3h, photos taken each 30s
Trajectory range: 18.75-600 m
Thrust level: $I_{sp} = 1500$ s with max thrust = 1.1 mN
Map level: 25 photos/face on **Rectangular** shaped object

In Fig. 4.13 the evaluation metrics of the training simulation are shown in terms of score, actor and critic networks' loss and entropy factor. It is important to understand that it is not possible to directly compare these values (in particular the *score*) to the discrete case's ones since the environment and reward definition are different.

The average map level results through the simulation are outlined in Fig. 4.14, where it can be noticed how the level is equal or higher with respect to the discrete case, e.g. the one Fig. 4.3. As already reported, this is due to a twofold reason: on the one hand, the map reconstruction may be easier to achieve

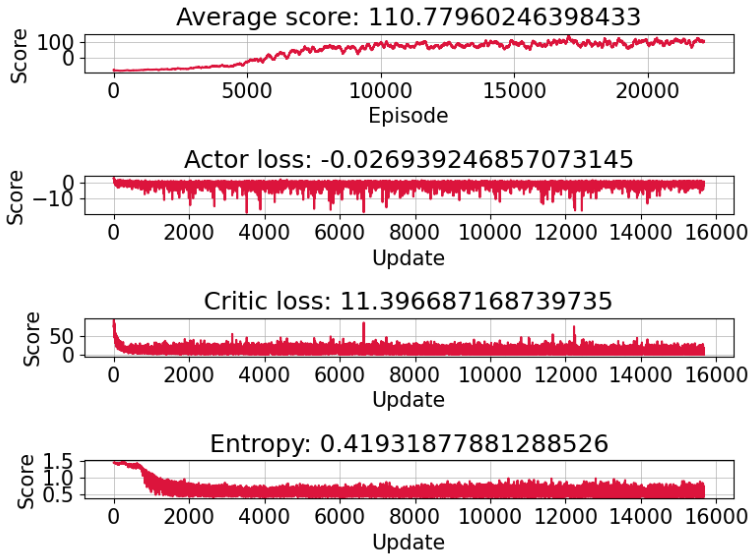


Figure 4.13: Continuos action space agent around a rectangular object: training simulation's evaluation metrics.

since the reward is shaped to consider a VIS-TIR camera; on the other hand, instead, the action is more complex to learn and use.

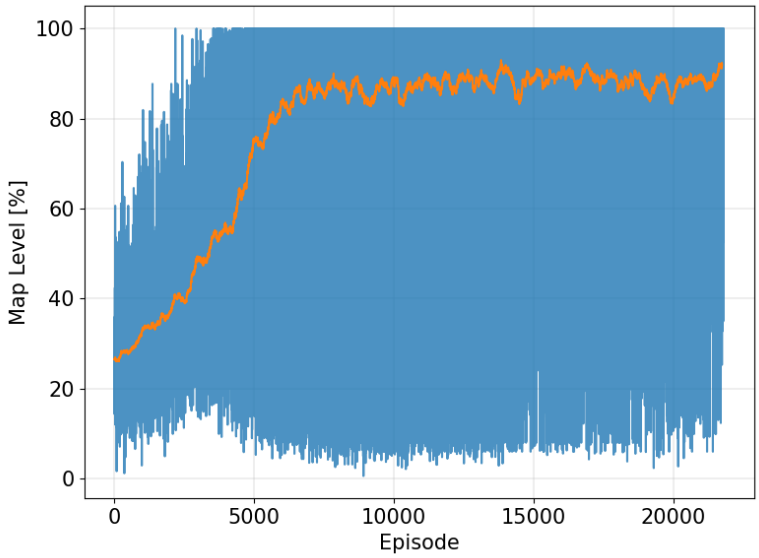


Figure 4.14: Continuos action space agent around a rectangular object: average map level during training.

VESPA case. Switching to the second case, i.e. VESPA, the environment is defined in the following green box. The resulting evaluation metrics and average map level profile during the training phase are reported in Fig. 4.15 and Fig. 4.16.

Continuos Space Agent - VESPA
Chaser-Target: rel. velocity fixed, position and attitude random
Camera FoV: 10°
Integration step: 1 s for max 3h, photos taken each 10s
Trajectory range: 50-500 m
Thrust level: I_{sp} = 1500 s with max thrust = 1.1 mN
Map level: 25 photos/face on **VESPA**

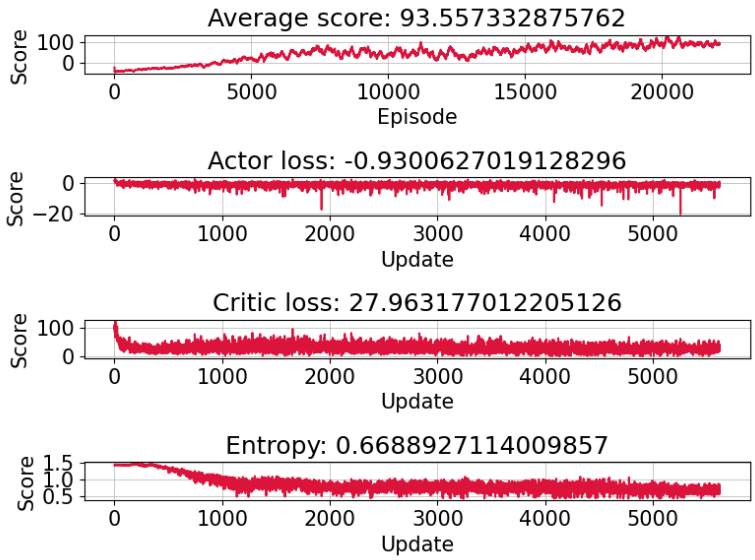


Figure 4.15: Continuos action space agent around VESPA: training simulation's evaluation metrics.

The results reflect the behaviour observed in the previous case. Focusing only on the map level's trend, here, the average value is slightly smaller. Two considerations can be made on this: the first concerns the shape of the object itself, which is quite different to the rectangular one, reason why the learning trend can be slower. The second is related to the dimensions of the object, in this case, VESPA is quite small with respect to the rectangle, and therefore this may affect the simulation.

Anyway, the outcoming performance level is high, peaking at about 95% of the covered map, so the training step can be considered successful; moreover, the

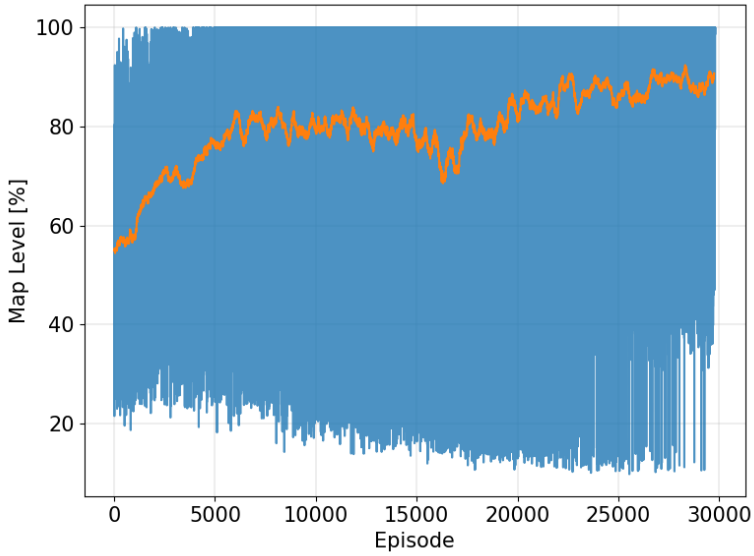


Figure 4.16: Continuos action space agent around VESPA: average map level during training.

profile of the average map increases over the span of the episodes and seems to be still improving, suggesting that a longer training could be beneficial. An example of trajectory around VESPA, which completes the 100% of the map, is shown in Fig. 4.17. On the right the evolution over time of the acceleration actions for the three axes commanded by the agent. Here, the feasibility of the action is guaranteed by constraints that avoid the acceleration to have a singularity in its evolution.

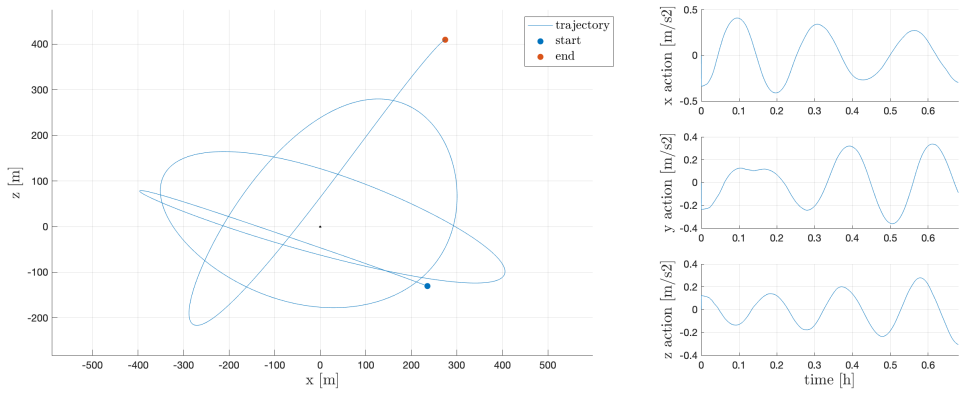


Figure 4.17: Continous agent around VESPA example trajectory.

To underline how the agent's training, whether discrete or continuous, is strongly influenced by the target object shape, dimension and also the defined mesh, in Fig. 4.18, another example of a trajectory of the spacecraft around the rectangular object. In this case, the relative path is longer because the dimension of the object and its relative mesh are much bigger than in the VESPA case and therefore the agent needs to prevaricate around the object to achieve the overall map.

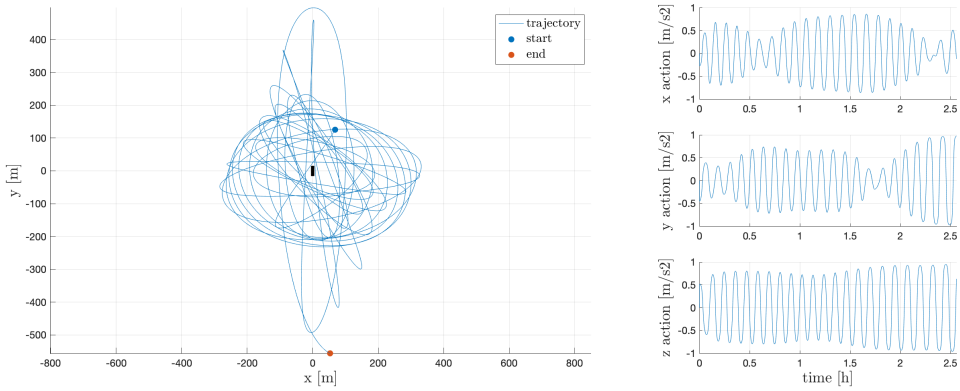


Figure 4.18: Continous agent around the rectangular object example trajectory.

To complete the comparison analysis between discrete and continuous cases, in Fig. 4.19 and Fig. 4.20, two trajectories performed by the discrete agent around VESPA and the rectangular object respectively are displayed. Even if in both cases the agent is capable of correctly achieving the objective without any consistent difference in the final result, the obtained flight path is more *linear* and controllable when failures occur in the continuous case.

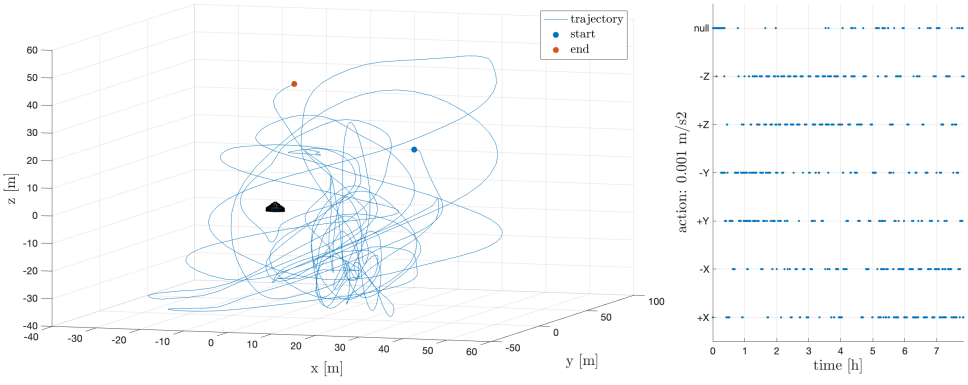


Figure 4.19: Discrete agent around VESPA example trajectory.

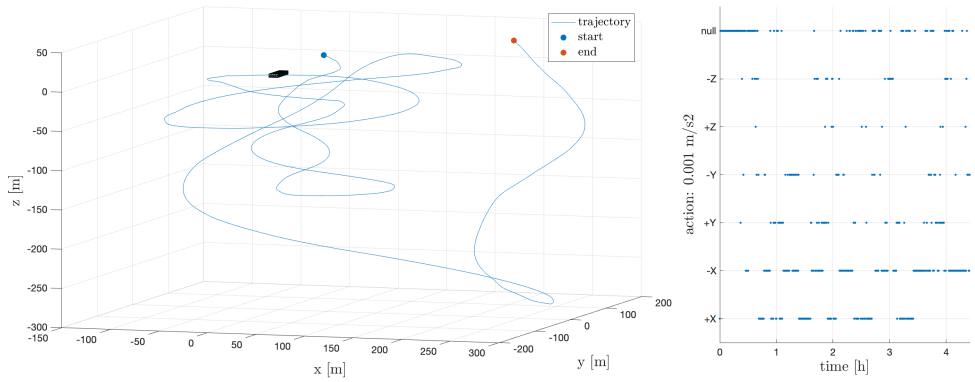


Figure 4.20: Discrete agent around the rectangular object example trajectory.

In conclusion, looking at the results obtained in both cases and comparing them to the baseline defined in Section 4.2, the effectiveness and feasibility of having a continuous action space is attested. Starting from this point, a sensitivity analysis must be performed to also assess the level of robustness of the model itself.

4.3.1 Sensitivity on the Continuous Action Space Agent

The sensitivity analysis on the continuous action space agent covers the following aspects related to previously unexperienced scenarios:

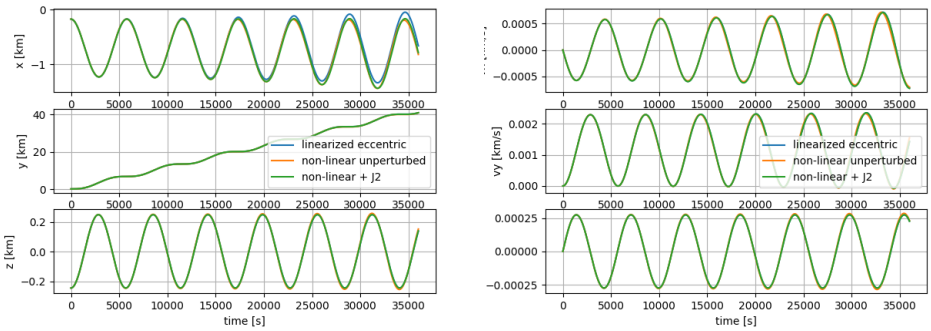
- Swap the linearized eccentric dynamics used during training with more complex nonlinear models, that should represent with more fidelity the real evolution of the relative motion between the chaser and the target. This test will be very useful to see how the agent behaves in a (even if *slightly*) different dynamical environment as it happens for the full AI-based GNC pipeline that will be developed in Chapter 5.

- Analogously to what will be treated in detail in the next Section 4.4, also random noise in the relative motion estimation is introduced and analysed. The pose is retrieved from navigation with the sensors on-board (in this case vision-based), which are affected by errors in their measurements.

- Sensitivity analysis on the rotational velocity of the target, carried out investigating the effects of a faster attitude motion.

Non-linear dynamics. Two nonlinear relative dynamics models are considered: unperturbed [71] and J_2 perturbed [72], whose difference with respect to the linearized eccentric dynamics employed during training, can be appreciated

in Fig. 4.21, where the free dynamics is propagated from the same initial conditions.



(a) Relative Position **(b)** Relative Velocity

Figure 4.21: Comparison between linearized eccentric and non-linear free dynamics.

It is worth noticing that the difference between the models is practically negligible for a time window wider than 5 hours. Since the episode may last only 3 hours at maximum, this suggests that the agent should be capable of performing well also when the dynamics are not the one it was trained on. This logic assumption is supported by the results in terms of average map obtained running the simulation tests and reported in Table 4.8.

	Map [%]
Linearized Eccentric	95.12%
Unperturbed Nonlinear	94.37%
J_2 Perturbed Nonlinear	94.57%

Table 4.8: Map percentage comparison between different models [69].

Navigation uncertainty. During the training and testing, the state variables were assumed to be correct and perfectly known. However, in a more realistic scenario, uncertainty is strongly present due to the errors, even if small, in sensor measurements. This paragraph aims to investigate what happens to the performance if noise is added at each time-step between the estimated value coming from navigation and the guidance block. Specifically, noise is added to the relative position and velocity vector components, sampling from Gaussian distributions defined by the following standard deviations: $\sigma_{pos} = 10m$ and $\sigma_{vel} = 0.1m/s$ (the same that will be used in the following section about *state uncertainty analysis*).

The first test is a simulation in which noise is applied to both variables and the performance level experiences a reduction, decreasing to about 80% of the average map level. The same test is performed by applying distinctively the two uncertainties, first on the relative position and then on the velocity. Table 4.9 summarizes the results of all tests.

Position + Velocity	Position	Velocity
80.38%	84.07%	87.58%

Table 4.9: Map percentage comparison with navigation uncertainty.

The model does not seem to be robust enough in this kind of scenario, so a deeper analysis is deemed necessary to better understand how uncertainty affects the performance level.

Target attitude analysis. The range for the starting target attitude motion was selected considering the conditions defined in Table 4.7. Here, a sensitivity analysis on the angular velocity is carried out, by comparing the results of two simulations: fast-slow target attitude. The *slow* case is the baseline continuous action agent case studied up to now, while the *fast* case enlarges the range from which the initial attitude motion gets sampled:

$$|\dot{\theta}_i| < 0.001\, rad/s \quad \longrightarrow \quad |\dot{\theta}_i| < 0.005\, rad/s$$

The principal model is then tested with this modification and the performance level drops down with respect to the nominal case to a performance level of only 69% of the map on average. Two are the main reasons that can be associated with this result: on the one hand, the state space is greatly augmented; on the other, the agent policy network is heavily influenced by the target rotational velocity, since the agent selects its next actions depending on how VESPA is rotating, to plan a trajectory that can inspect the faces it has yet to see.

To solve for this issue, a new model training is set up, keeping the same architecture and parameters used before and simply enlarging the rotational velocity range to the one employed during the test.

Thanks to this new training phase the agent can improve its performance level, reaching about 80% on average of the map reconstructed, as visible in Fig. 4.22. However, the performance level is lower than the one obtained by the principal model, but this is expected: indeed, the state space has been greatly expanded, so a training procedure with the same number of episodes will inevitably bring worse results. Nevertheless, the agent is still capable of learning a quality policy and if trained for a higher number of episodes, the result would benefit and reach higher levels.

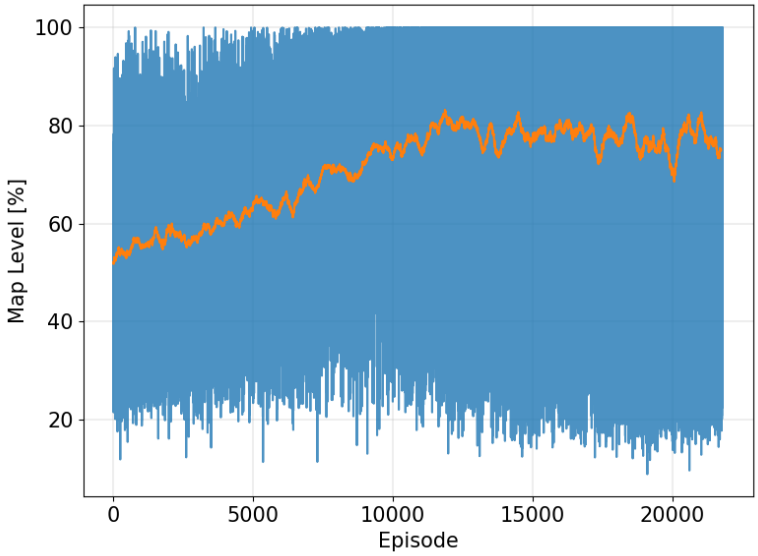


Figure 4.22: Average map percentage during fast attitude training.

Therefore, a faster target rotational dynamics does not seem to be a bottleneck for the model performance, but rather this extension of the state space simply makes the required training step longer.

4.4 State Uncertainty Analysis

A faulty assumption employed during both the training and testing phases of the proposed guidance algorithm is the correct estimation of the relative pose with respect to the target at each time step. In reality, these estimations are, in most cases, the output of an estimation filter, which takes as input the model of the dynamics and the sensor measurements, that are inherently affected by noise and imperfections. This means that the assumption of a perfectly known state is not satisfied a priori and, as such, a deeper analysis is required to verify the applicability of the algorithm in higher fidelity scenarios.

Here the autonomous guidance performance is evaluated in the presence of noise, which, as just said, can be a result of both errors in the sensor measurements or in the mathematical formulation of the problem, which may not be able to represent the overall characteristics of the environment. In this analysis, noise is added at each time-step between the estimated value coming from navigation and the guidance block. Regarding the problem at hand, noise is assumed to affect the state input vector in terms of chaser-target relative position and velocity, as well as the attitude angles describing the relative target rotation. As specified in the Section 3.1, the incoming state vector is considered to

result from an image-based navigation system. Therefore, much attention has been paid in order to properly define the type of errors affecting the navigation outputs. Indeed, most of the time, image-based navigation systems are not affected by deterministic errors, but by stochastic errors. For this reason, concerning the relative position and velocity, a uniformly random error constrained in predefined ranges is taken into account. Differently happens for the angular position error, where Gaussian noise is applied to the input state, as motivated in [73]. The error values used in the following simulations are outlined in Table 4.10. These are typical values for the VIS image-based navigation tool, as the one proposed in [64]. In Table 4.10, also the standard deviation considered to describe the angular error is defined: it is valid for all three cartesian directions.

	Type	Min	Max
Position Error	Uniform	-10 m	10 m
Velocity Error	Uniform	-0.1m/s	0.1m/s
	Type	Mean	SD
Attitide error	Gaussian	0	2°

Table 4.10: State input errors types and values.

In the following, the results of the sensitivity analysis are generated starting from a discrete action space agent, as defined in Section 3.1.2. The results are focused on a target object shaped as a simple rectangular, exactly as the one defined in Section 3.2, and shown in Fig. 3.2. The environment characteristics are summed up in the following box.

State Uncertainty
Chaser-Target: rel. position/velocity fixed, attitude random
Orbit features: Sun phase random, true anomaly θ random
Camera FoV: 10°
Integration step: 30 s for max 86400 s
Trajectory range: 18.75-600 m
Thrust level: 0.001 m/s^2
Map level: 25 photos/face on **Rectangular** shaped object

The target object's orbit initial conditions are the same ones defined in Table 4.4.

4.4.1 Sensitivity Analysis

The first analysis compares the performance level (in terms of mapping level) of the trained agent with nominal input state and with noisy input state, testing

it for 5000 episodes. In Fig. 4.23, the mapping level of the trained agent with noisy input in terms of relative position is shown.

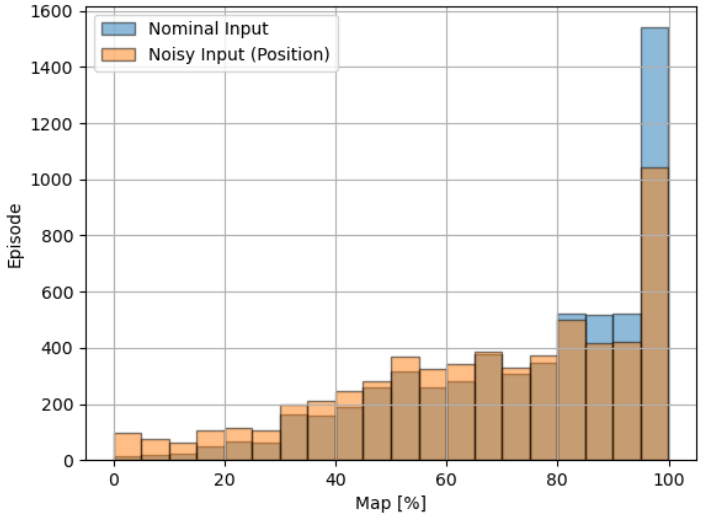


Figure 4.23: Map level comparison between the trained discrete agent with nominal input and with noisy chaser-target position input (rectangular object).

The histogram shows that the overall agent's performance lowers down, but not dramatically; on the contrary, it happens when the navigation errors are applied also to the relative velocity and attitude position, as shown in Fig. 4.24 and Fig. 4.25 respectively.

Moreover, the fact that the sensitivity analysis on the angular velocity error was not performed may seem strange. The reason is twofold: on the one hand, the agent was already trained in an environment with random angular velocity, on the other, recalling the Euler's equations that drive the target attitude, thanks to the linearization, the velocity directly depends on the angular position, therefore the agent should already be aware of how the attitude dynamics works; thus the angular velocity error is not expected to influence the performance more than the angular error. Fig. 4.26 shows the result derived by testing the trained agent with all the noisy input states, tracing the trend of the worst error due to angular position uncertainty. As it is, the agent is completely useless for a real image-based navigation system. The same analysis is also performed for an LSTM agent model as displayed in Fig. 4.27, in which similar results can be observed.

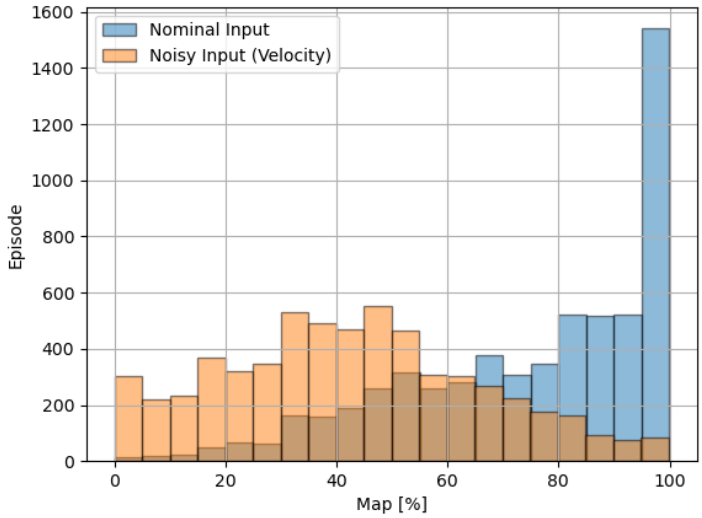


Figure 4.24: Map level comparison between the trained discrete agent with nominal input and with noisy chaser-target velocity input (rectangular object).

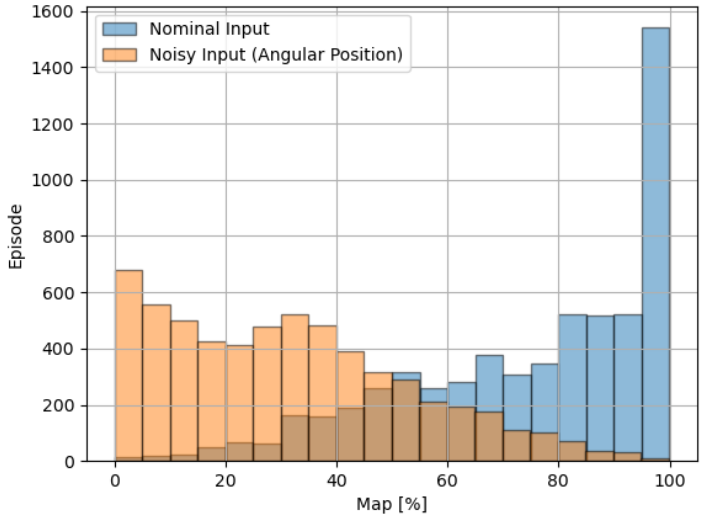


Figure 4.25: Map level comparison between the trained discrete agent with nominal input and with noisy chaser-target angular position input (rectangular object).

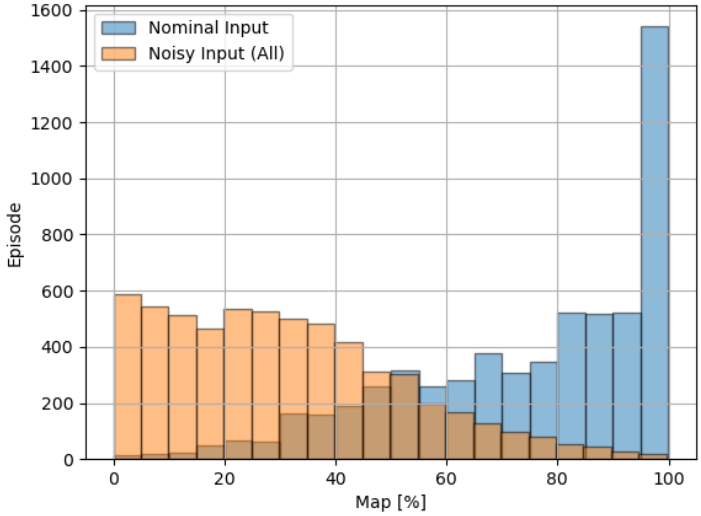


Figure 4.26: Map level comparison between the trained discrete agent with nominal input and with overall noisy input (rectangular object).

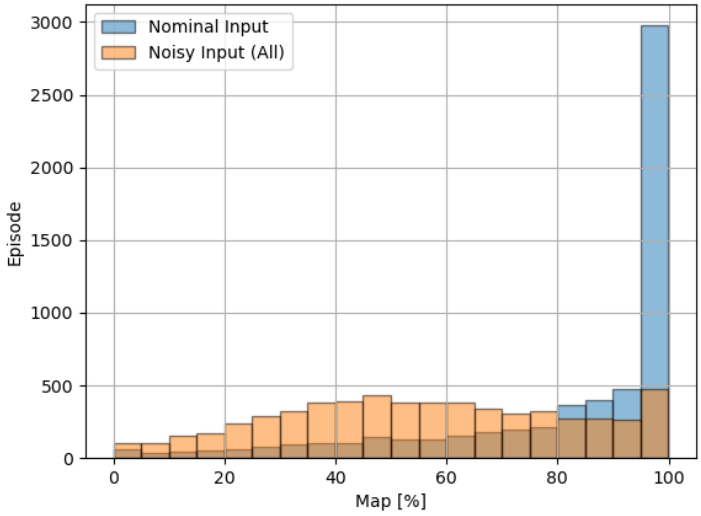


Figure 4.27: Map level comparison between the trained discrete agent (LSTM model) with nominal input and with overall noisy input (rectangular object).

Moreover, to increase the characterization and understanding of the robustness properties of this discrete agent model, the same analysis performed on the

general-purposed rectangular object is carried out also for a different target object shape, namely the VESPA debris, depicted in Fig. 3.3. Similarly to before, in Fig. 4.28 the comparison between the trained agent (this time on VESPA) fed by nominal and noisy inputs is shown. It is notable that, with this object, the performance really drops down, underlining how the efficiency of the agent model is different in terms of objects or state error sensitivity. Indeed, even if the trained agent performs better around VESPA than with the rectangular object (see previous section Section 4.3), it is greatly more sensitive to noisy input.

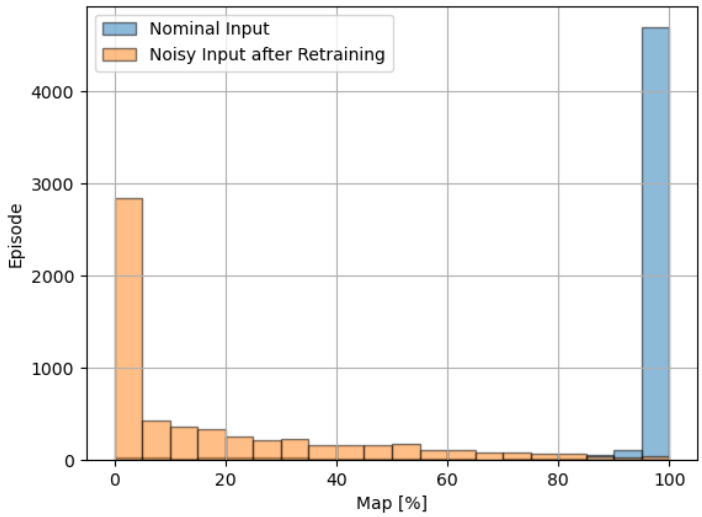


Figure 4.28: Map level comparison between the discrete agent with nominal input and with overall noisy input (VESPA object).

In conclusion, it is quite easy to notice how state uncertainty can be a strong problem for this kind of autonomous agent. To overcome this issue, two possible approaches have been analysed: re-training the neural network by adding noise to the state input or applying TL and then training the agent with noisy input. The comparison between fine-tuning and transfer learning is deeply analysed in different reinforcement learning fields of applications [74, 75].

In particular, fine-tuning involves training a pre-trained model on the target task with a smaller learning rate [76]. In DRL, fine-tuning can be mainly performed in two ways:

1. **Policy Fine-Tuning**: fine-tune the policy network and, if necessary, adjust the neural network architecture.

2. **Hyperparameter Tuning**: fine-tune hyperparameters like learning rate, discount factor (γ, λ), and exploration strategy (e.g. with entropy level).

On the other side, transfer learning involves using a pre-trained neural network as a starting point for a new task [77, 78]. TL can be realized through various methods such as:

1. **Feature Extraction**: transfer the lower layers of a pre-trained network (e.g., convolutional layers in a CNN) and train only the upper layers specific to the new task.
2. **Policy Transfer**: transfer the policy network trained on a source task to the target task and fine-tune it.

Fine-tuning and transfer learning are not always at odds and can also be used together; starting from the major advantages and disadvantages of the two approaches summarized in Table 4.11, the two methods can be leveraged and evaluated based on the following factors:

- **Task Relatedness**: TL is suitable when the source and target tasks are related while fine-tuning can be applied to any task.
- **Training Efficiency**: TL is generally more time-efficient as it starts with pre-trained weights, while fine-tuning from scratch may take longer.
- **Adaptability**: Fine-tuning excels in tasks requiring task-specific adaptations and customization.
- **Data Availability**: TL can be data-efficient, making it a good choice with limited data, while fine-tuning might require more data.
- **Risk Tolerance**: TL may carry the risk of negative transfer if the source task is too dissimilar while fine-tuning avoids this risk.

In summary, transfer learning is advantageous when dealing with related tasks and limited resources, as it speeds up training and enhances generalization. Fine-tuning, on the other hand, offers greater adaptability and task-specific optimization, making it a better choice when the source and target tasks are closely related or when fine-tuning from scratch is required. In the end, in practice, the choice between transfer learning and fine-tuning can be only based on the specific requirements and constraints of the particular DRL application.

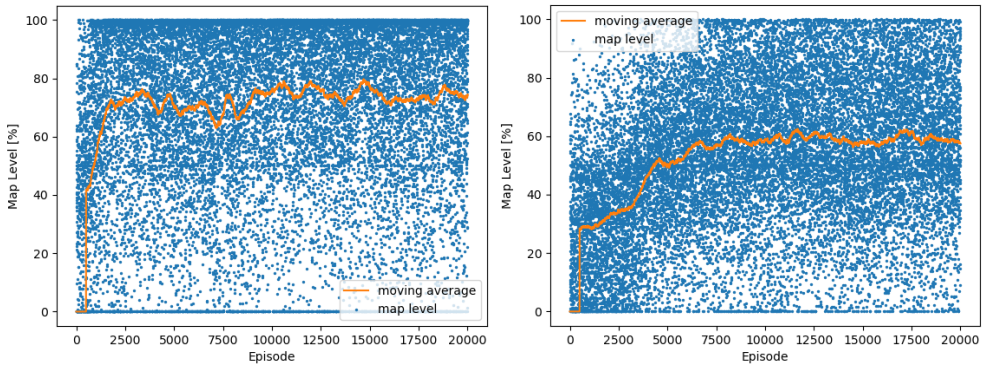
In the following two sections, both approaches will be exploited and analysed.

Fine Tuning			
Advantages		**Disadvantages**	
Adaptability	Fine-tuning allow the model to adapt more closely to the target task's requirements, making it suitable for tasks with specific constraints.	Slower Convergence	Fine-tuning from scratch may require more training time compared to transfer learning.
Task-Specific Optimization	Fine-tuning permits task-specific optimizations and customization.	Potential for Overfitting	Fine-tuning can lead to over-fitting if not carefully managed, especially on small datasets.
No Source-Target Dependency	Fine-tuning can be applied to any target task without the need for a related source task.	No Source-Target Dependency	Fine-tuning can be applied to any target task without the need for a related source task.
Transfer Learning			
Advantages		**Disadvantages**	
Efficient Pre-trained Knowledge	Leveraging pre-trained weights can significantly reduce training time on the target task.	Source-Target Mismatch	The source and target tasks should be related; otherwise, transfer learning may not be effective.
Generalization	Transfer learning allow the model to generalize better as it inherits knowledge from the source task.	Limited Adaptability	The pre-trained model may not adapt optimally to the specifics of the target task.
Data Efficiency	It often requires fewer samples to adapt to the target task due to prior knowledge.	Risk of Negative Transfer	In some cases, transferring knowledge may harm the performance of the target task if the source task is too dissimilar.

Table 4.11: Advantages and disadvantages of transfer learning and fine-tuning techniques for deep reinforcement learning.

4.4.2 Re-training Technique

The re-training phase has been performed coherently with the training analysis already done, meaning that, even if the agent is trained from scratch, the way in which it is processed, all the hyperparameters (except for the *learning rate* that needs be tuned in any way) and properties remains unchanged. Therefore, the agent is re-trained, but this time the erratic input information becomes part of the training simulation. These have been carried out for both the feed-forward network (MLP) and the recurrent one (LSTM).



(a) Nominal agent (MLP). **(b)** Re-trained agent (MLP).

Figure 4.29: Map level training trends comparison between nominal MLP agent and the re-trained MLP agent.

Fig. 4.29 displays and compares the map level training curves of the nominal and re-trained agents for the MLP case. It can be noticed that the map level trend for the re-trained agent sets at a lower level with respect to the baseline on the same simulation length; indeed as reasonable, the state space now is much bigger than before due to the addition of random pose uncertainties and, therefore, more difficult to explore and experience. Nevertheless, the results, which can be appreciated in Fig. 4.30, show anyway good and recovered performances; in fact, the training architecture model is confirmed to be robust to environmental changes. Moreover, thanks to the addition of the noisy input directly in the learning loop, the agent is capable of recovering its previous performance level. The same procedure is held also for the LSTM case: in Fig. 4.29 the map level training curves of the nominal and re-trained agents are compared. The behaviour detected is quite similar to the one obtained for the previous re-trained case, thus again the maximum map level reached by the new agent is lower with respect to the baseline one. Moreover, being the LSTM a more complex neural network than the feed-forward model, achieving the same performance level may require longer training: indeed, due to a

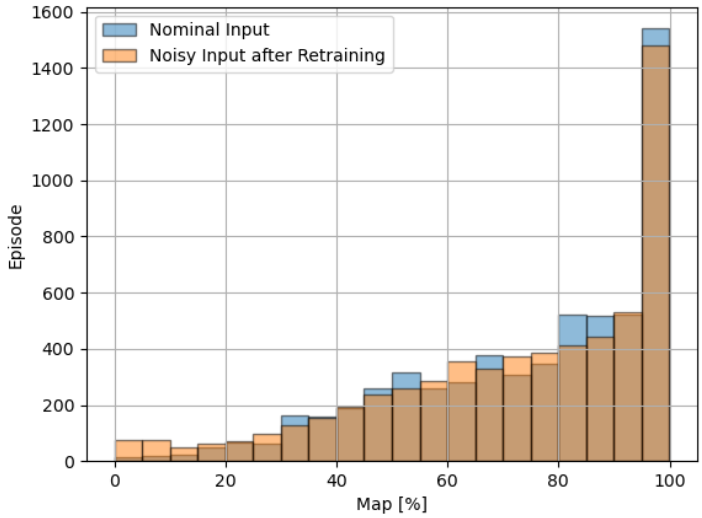
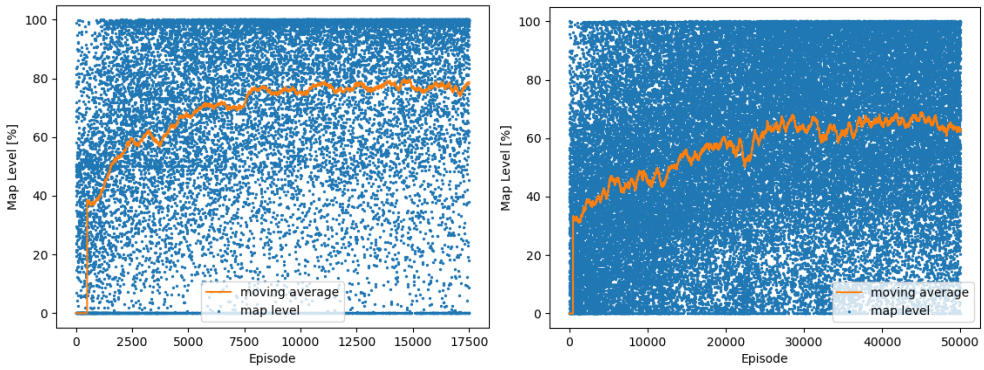


Figure 4.30: Map level comparison between trained agent (MLP) tested with nominal input and re-trained agent (MLP) with overall noisy input.

much bigger state space, the simulation time was higher (50000 episodes), with respect to the nominal or MLP ones.



(a) Nominal agent (LSTM). **(b)** Re-trained agent (LSTM).

Figure 4.31: Map level training trends comparison between nominal LSTM agent and the re-trained LSTM agent.

Differently from the previous case, from the results showed in Fig. 4.32, the re-training technique does not seem to be so effective on the LSTM model. Even if the average level is higher with respect to the nominal LSTM agent affected by input uncertainties, in Fig. 4.27, the performance is not good

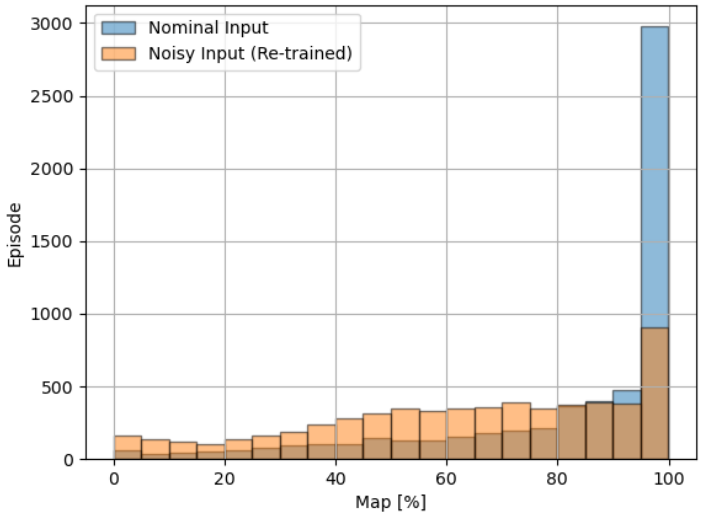


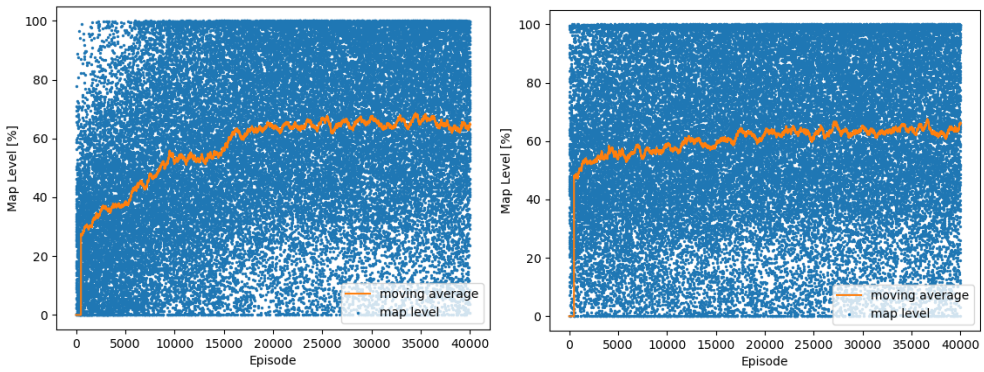
Figure 4.32: Map level comparison between trained agent (LSTM) tested with nominal input and re-trained agent (LSTM) with overall noisy input.

enough to consider the errors negligible due to the uncertainties. This is the main reason behind the choice of exploring another methodology in order to enhance the robustness of this agent.

4.4.3 Transfer Learning Technique

As introduced before, transfer learning is an advanced machine learning method that exploits an already trained model to perform a specific activity as a starting point for the development of a new model intended for the execution of a second or different activity. In the specific case here proposed, another training phase is built on the two networks already trained with the nominal *perfect* input, aiming to tune the models and increase their robustness to state uncertainties. In Fig. 4.33, the learning curves in terms of map level are shown: the MLP in Fig. 4.33a and the LSTM in Fig. 4.33b. As happened before in the re-training analysis, the level achieved is high but not equal to the one reached by the nominal input case also for the transfer learning. The reason is always related to the increase of the state space.

The resulting performances are presented as histograms in Fig. 4.34 and Fig. 4.35, where the comparisons between nominal and transfer learning for MLP and LSTM models are shown respectively. The outcomes are in line with the re-training results, so at first sight, it may seem that both the two methods obtain more or less the same results, thus, they are in some way interchangeable.



(a) MLP agent with transfer learning **(b)** LSTM agent with transfer learning

Figure 4.33: Map level training trends of noisy input trained agent with transfer learning.

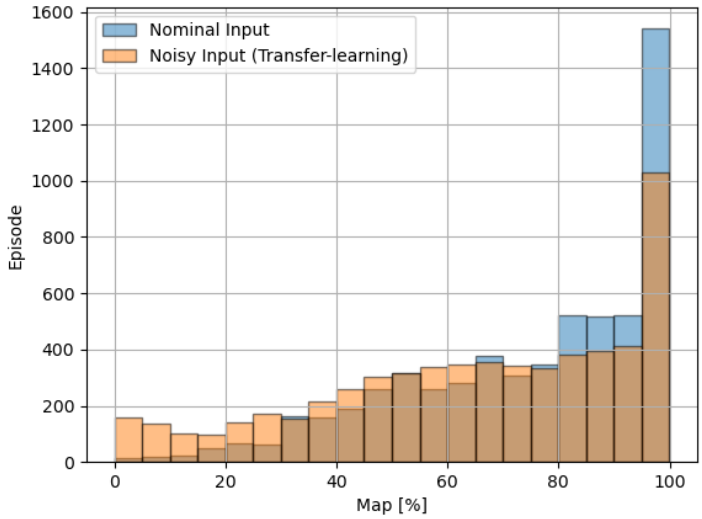


Figure 4.34: Map level comparison between trained agent (MLP) tested with nominal input and TL-trained agent (MLP) with overall noisy input.

On the contrary, the results showed in Fig. 4.36 and Fig. 4.37 demonstrate that this superficial outcome is not strictly applicable, because, when performing a sensitivity analysis on the state input vector of the re-training and the transfer-learning agents fed by noiseless input, the latter responds much better than the former, since it still maintains knowledge about the *perfect* input state training. This makes us believe that applying TL gives a higher level of robustness also considering other levels of noise, while re-training must be tuned every time

the input state characteristics change. In conclusion, TL seems to be the most reliable technique to face state uncertainty.

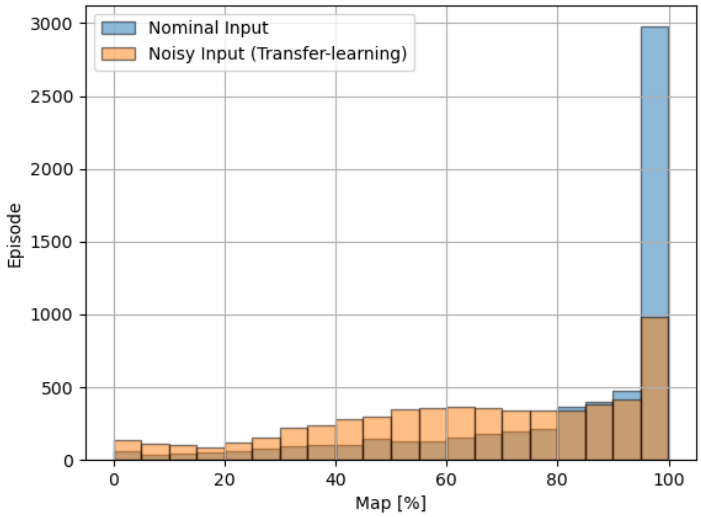


Figure 4.35: Map level comparison between trained agent (LSTM) tested with nominal input and TL-trained agent (LSTM) with overall noisy input.

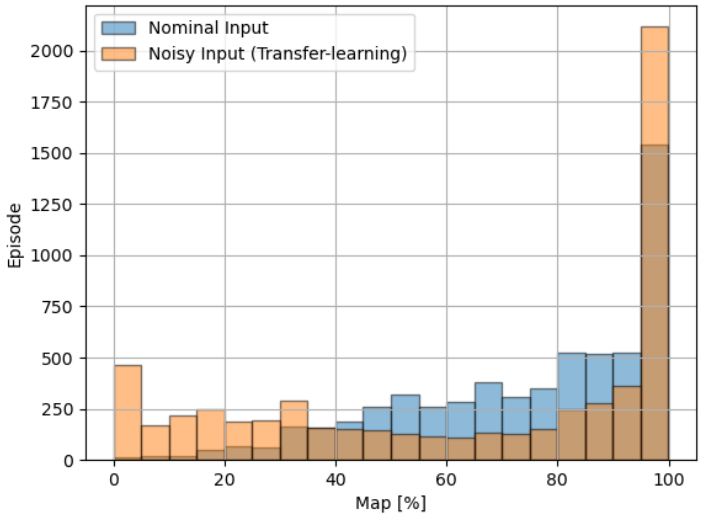


Figure 4.36: MLP: map level comparison between trained agent tested with nominal input and TL-trained agent with nominal input.

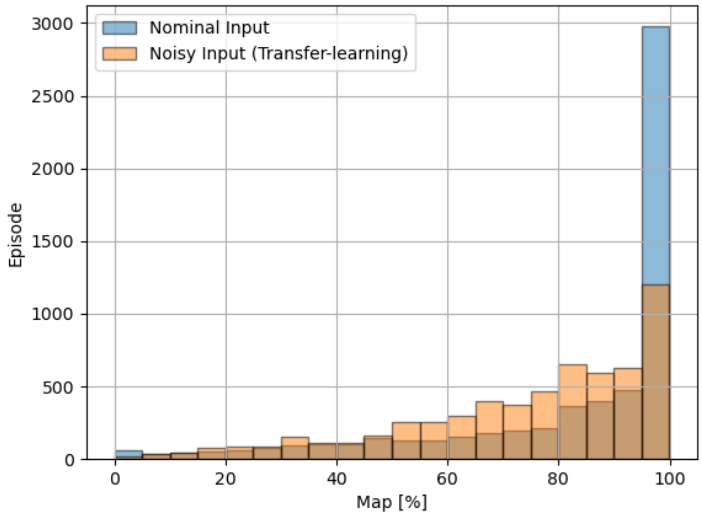


Figure 4.37: LSTM: map level comparison between trained agent tested with nominal input and TL-trained agent with nominal input.

4.5 Validation Analysis

To assess the real potential of this kind of process it is necessary to evaluate the proposed method's performance with respect to more classical guidance approaches. Before doing that, it is also important to compare the results to benchmark cases in a dedicated analysis.

Benchmark. This paragraph reports and sums up the tests carried out to assess the performance of the models discussed up to now, against some simple benchmarks, to confirm the effectiveness of the learning step and the reward function design. Even if it may seem trivial, the first two comparisons are against *no-learning* models, meaning that they have not gone through the training procedure: the first simply propagates the free-dynamics starting from random initial conditions; the second is, instead, a random control model. The average map percentage obtained by both models is reported in Table 4.12 and comes from the scenario characterized by the continuous action space agent around VESPA.

Free-Dynamics	Random Control	Trained Agent
52.9%	55.5%	95.5%

Table 4.12: Comparison of benchmark models map percentage.

Therefore, the nominal model performs much better than both of the others two, verifying the training effectiveness. A further benchmark test is performed by comparing the performance with a model that undergoes the training step, but with a simpler reward function, entailing just the chaser-target distance objective. As such, the agent learns how to remain in proximity of the target, keeping itself in the safe region of space, but it does not learn how to map it efficiently, because no information regarding the map level and the quality of images is fed to its policy network. This kind of analysis reflects the one carried out in [26] for the A2C agent. This *simple* model performs worse than the nominal one (reaching about 73% of the average map level), confirming the good design of the reward function, which incentivizes the agent to better perform the shape reconstruction. Moreover, since its main objective is to simply remain inside the boundaries in space, it takes much longer, on average, to complete the map, settling for an increment of almost 100% (3150s against 1595 of the nominal model). In conclusion, the nominal model takes nearly half the time to cover 100% of the target map, thus confirming that it has learnt a different, more efficient strategy for mapping VESPA, than simply remaining inside the limits.

Validation. As a validation method, the proposed algorithm is here compared against the performance level obtained from the e.Inspector mission project. The design of e.Inspector mission trajectory aimed at VESPA inspection is defined in [64] based on a more classical approach regarding GNC and relative dynamics, so it is interesting to juxtapose it to an AI-based architecture.

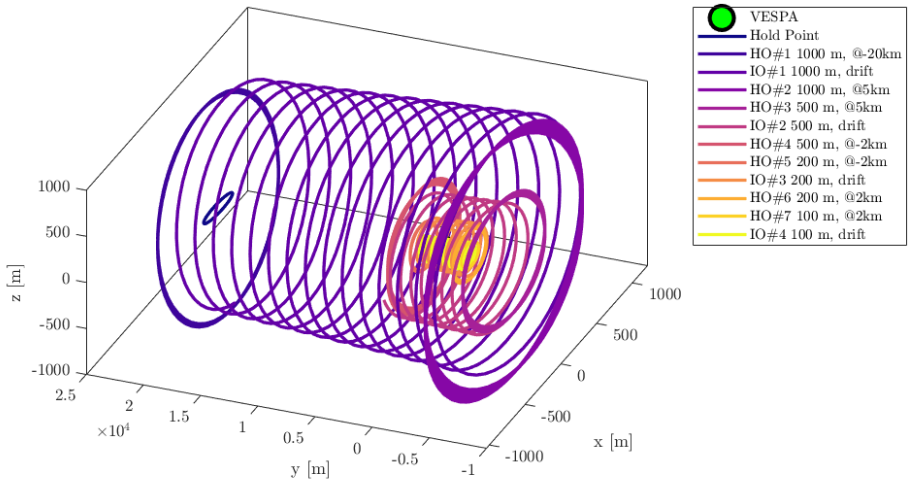


Figure 4.38: Image phase of the e.Inspector trajectory mission [64].

The e.Inspector ESA project is currently in Phase-B and its goal is to fly a small satellite around Vespa to inspect it and enhance the target knowledge. The trajectory is robust and passively safe as displayed in Fig. 4.38 and is designed to have many hold orbits and inspection trajectories, which last for almost a month.

The interesting part of the trajectory is the one which remains inside the boundary limits defined for the learning agent training, so between 50 m and 500 m of relative distance with respect to the target. The integration time-step used in the original simulation is 60 s, so to be compatible with the work of this thesis, a preliminary interpolation of the trajectory is carried out to augment it such that the time elapsing from one point to the next is 1 s, thus equal to the time-step used for the continuous action space agent. Analysing the resulting trajectory with the same reward function used for the nominal model, a couple of notable points are highlighted:

- The duration of the free-drifting trajectory is about 6.5 hours, which is more than enough to complete the map reconstruction;
- The only variable feature in the simulation is the target attitude evolution, which is initialized randomly. The trajectory never changes between different runs, since no control action is actuated and the starting point is always the same.

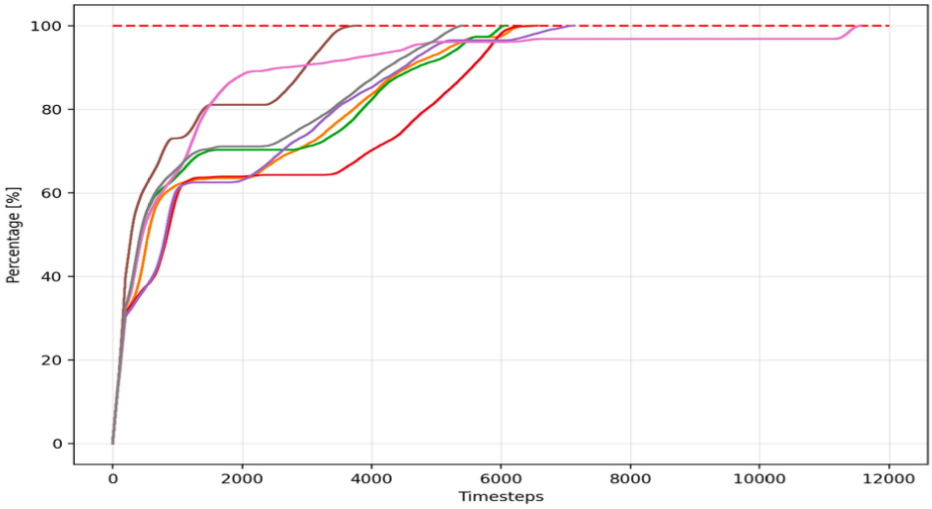


Figure 4.39: VESPA map reconstruction evolution of different simulations exploiting the e.Inspector trajectory [66].

A visual representation of the map evolution during different simulations is reported in Fig. 4.39. Note that at the beginning the profile is the same for each run and then it diverges depending on the attitude dynamics of the target.

The simulations are stopped when the performance level reaches the one of the nominal model, or, alternatively, when the covered map percentage reaches 95%. To account for the target attitude randomness, 5000 tests are performed and the result is simply given by the average. The average number of time steps to acquire a map coverage of at least 95% is around 1.44h. On the contrary, the nominal model simulations last way less, around 0.43h. This outcome has two main implications: the first is related to the fact that time is a particularly precious resource in space activities, and speeding up the mapping operation allows to perform other duties with greater flexibility. On the other hand, instead, a free-drifting trajectory will always be more convenient in terms of consumed propellant, but if the fly-around lasts for a few minutes, then the consumption is reduced to the minimum necessary.

In conclusion, the DRL-based approach may be considered with its pros and cons as a valuable alternative with respect to more classical guidance design, for the moment at least from a theoretical point of view.

4.6 Closing Remarks

In this chapter, an in-depth analysis of the innovative autonomous guidance algorithm for the shape reconstruction of an uncooperative space object, developed via Deep Reinforcement Learning has been presented. Starting from the reference point set in [26], the method has been refined, by comparing the performance obtained with different neural network architectures, training the agent with multiple rewards and different random initial conditions. Specifically, it has been found that the exploitation of RNN architectures improves training stability and reduces oscillations during the learning phase. Although the same level of performance is, however, achieved by the simpler FFNN models, the robustness showed by the RNN is significantly better.

A further important step forward in the investigation of PPO for solving the spacecraft decision-making policy is made by implementing the algorithm with a continuous action space, in order to simulate more realistically the actuators' control on the chaser motion. The two models, *discrete* and *continuous* action space, are then compared and evaluated in the same scenario, and the result is critically commented. It is, therefore, shown that at the cost of a more complex and longer training process, the continuous agent reaches, and sometimes overcomes, the performances of the discrete agent. Afterwards, the continuous action space model is extensively tested to asses its performance, robustness and sensitivity against unseen conditions, underlining its strengths and weaknesses.

Once the more realistic action space is specified, also the uncertainty on the state space has been analysed. The nominal autonomous agents, trained with again the two different network models, FFNN and RNN, have been

extensively tested to assess the model robustness and sensitivity, aimed at specifically analysing the case of uncertainty in the state and how it affects the performance of the algorithm. The first results showed how the main models are not robust to the noisy input state. Therefore, to overcome this problem, two different methods have been proposed and analysed: a re-training procedure and a transfer-learning procedure. Both of them improve the capabilities of the autonomous guidance agent in terms of average target mapping when uncertainties are present in the state input. Nonetheless, it is demonstrated that the transfer-learning method is more robust to further change in the input noise.

Some possible improvements can be discussed to enhance the performance of the proposed method. Indeed, beyond these two procedures, it may be beneficial to improve the deep reinforcement learning model with the introduction of a model-based network. Moreover, one of the main drawbacks of the agent, in its current form, is its inability to consider the chaser attitude, assuming that it is always pointing toward the target. Modelling, then, the chaser attitude dynamics would bring the discussed method much closer to reality, making sure that the map level increases only when the chaser is correctly pointing the camera and can take pictures of only the faces that are inside the line of sight and with the correct illumination conditions.

At the end of the chapter, the methodology is verified into a benchmark and validation analysis, which has been carried out to support the power and strengths of DRL-based approach, critically evaluating its performance with respect to more classical guidance approaches. In conclusion, these outcomes place, with confidence, the proposed approach into the valuable alternative pool for spacecraft guidance design.

CHAPTER 5

Autonomous AI and Image-based GNC Scenario

> Nothing good happens after 2:00 am, so when 2:00 am rolls around, just go home and go to sleep.
>
> — Ted Mosby, *How I Met Your Mother*

In this chapter, the AI and Image-based GNC algorithm will be described and its results will be presented. At first, in Section 5.1 the overall GNC pipeline is introduced and the new players involved, i.e. IP pose estimation tool and image-based navigation filter, are briefly explained (being not active part of the thesis work, it is not necessary to go into too much detail). Also, a very brief overview of the POV-Ray based IG tool is here outlined. In Section 5.2 the guidance and control tool is trained and extensively tested in order to optimally suit the new environment, characterized by the TANGO object. Afterwards, the GNC pipeline is tested in Section 5.3 following a simulation plan which increases the complexity of the environment and the tool step after step.

5.1 GNC pipeline Description

The AI and image-based GNC algorithm is based on four different blocks:

- **Dynamics.** This block defines the chaser-target relative environment: the relative position and velocity are computed from the absolute position/velocity values of the two integrated in ECI reference frame with J2 disturbance. This dynamics has been defined in Eq. 2.17. The relative attitude dynamics is derived from a quaternion-based integration in ECI reference frame.

- **Image Generation** and **Image Processing.** These two steps have been developed in [79], and taking the input of the dynamics in ECI frame, generates the relative target position (r) and attitude (q) with respect to the chaser camera frame, which, in this analysis, corresponds to the chaser body reference frame.

- **Navigation.** The H-infinity navigation filter is a linear filter for relative translation estimation based on two main assumptions: the input measurement is the target relative position in the chaser LVLH frame; the dynamical model is a linearized J2 model with control input. Therefore, assuming the knowledge of the absolute state of the satellite (reasonable if considering missions with at least one GPS or star tracker), it is possible to rotate the output of the IP block into the chaser LVLH reference frame required by the filter. At the end, the filter gives an estimation of the relative position and velocity in the chaser LVLH frame.

- **Guidance** and **Control.** This last block is based on the work done in this thesis and requires as input the relative chaser-target position and velocity in target LVLH frame, and the relative attitude of the target with respect to the chaser camera frame. Although these quantities are not directly available from the navigation filter or the image processing estimation, their outputs can be manipulated in order to get the correct input information. These *manipulations* will be explained in the next section. Finally, the block outputs the control action of the chaser (shaped as an acceleration vector) that affects both the dynamics and navigation blocks.

In Fig. 5.1 the algorithm pipeline is depicted: it can be noticed how the relative attitude required by the guidance block may be derived from adding Gaussian noise to the real dynamic value or directly from the IP block. This will be one of the steps performed to increase the complexity of the simulation during the testing analysis.

All of the blocks involved need an initialization process that shapes them in the correct environmental conditions based on `.yaml` input files. Each of the

blocks has its own evaluation metrics: in this thesis the ones of the guidance and control block will be mainly analysed and discussed.

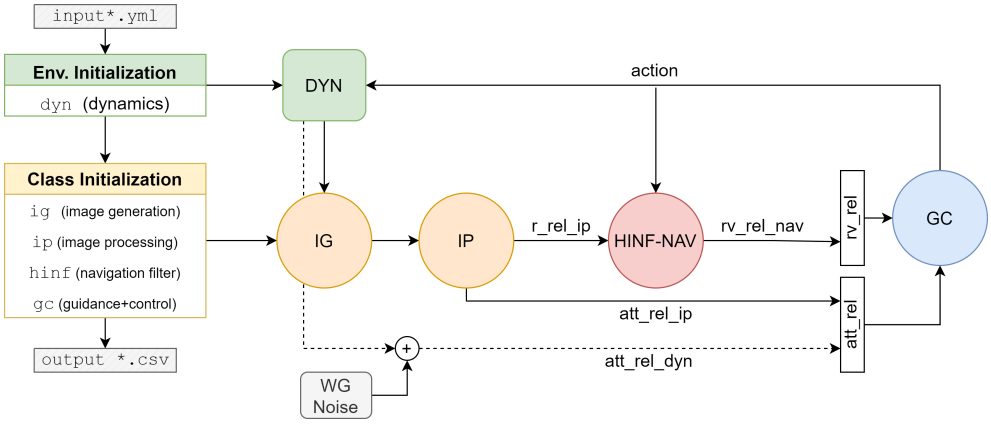


Figure 5.1: AI-based GNC pipeline algorithm scheme.

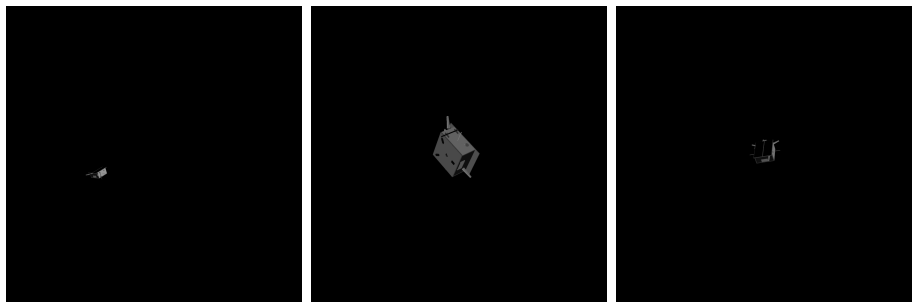
5.1.1 Image Generation

In recent years, new image generation methodologies have grown and increased their importance in a sector where more and more vision-based algorithms are developed and can not be tuned, due to the lack of publicly available spaceborne image datasets captured during flown missions. The scenario treated in this thesis required an Image Generation Tool for the VIS range, in particular, the used one is based on POV-Ray [80] and was developed in [62].

POV-Ray is one of the state-of-the-art software for image generation in space applications, in particular, it has already been employed to render via ray tracing natural landscape, e.g. Moon's surface [81, 82], and spacecraft [83, 62].

The IG tool is validated by means of quantitative indexes [84] making a comparison between the newly generated dataset [85] and the SPEED dataset as reference [86], whose validation was achieved comparing the synthetic images with respect to the real ones [87]. This tool adopts a high-fidelity noise model based on the photon detector model prescribed in [88] and implemented in [89, 90]. For a complete overview of the IG method, please refer to [79].

In Fig. 5.2 some examples of the rendered TANGO images are depicted, taken in different moments of one of the testing simulations. Although the IP performance is not affected by the Earth or atmosphere, neither of them has been used, and therefore, the black space has been considered always in the background. Since the GNC algorithm is in a closed loop, this background constraint is assumed because the rendering of the image with Earth and atmospheric models would have heavily increased the length of the simulation.



(a) Frame - 0 **(b)** Frame - 1 **(c)** Frame - 2

Figure 5.2: Samples of TANGO rendered images taken from a testing simulation of the GNC pipeline.

Input	The IG block requires the following quantities: chaser and target position in ECI reference frame, chaser and target body orientation in ECI reference frame, Sun position in ECI reference frame.
Output	TANGO image based on the Chameleon 3 camera model.

Table 5.1: Input/Output of IG block.

5.1.2 AI-based Space Object Pose Estimation

The IP block is a keypoint-based pose estimation algorithm, which leverages on CNN-based keypoint detection methodology, widely analysed also for space applications [87, 91]. The pose estimation algorithm used in this GNC pipeline relies on CNN trained to learn the specific position of each keypoint in the input image, similarly to the model adopted in [73]. The chosen model is the YOLOv8-pose model[1] which was born to bridge the synthetic image generation world to the mock-up image generation one [92]. This model is able to perform target detection and keypoints regression through a single inference and is characterized by a low computational effort. It was firstly tested on the famous SPEED dataset [93] and tuned for [94, 95]. For a complete overview of the algorithm in terms of architecture, training and performance please refer to [85]. Two main features and constraints must be cleared before entering into the detail of the GNC pipeline testing in Section 5.3:

- The IP model is tuned on a TANGO-based scenario, which then is affected by the object dimensions. In particular, TANGO's maximum length is more or less 1m, this means that, considering the Chameleon 3 camera features already defined, the chaser can not drift apart of more than

[1] https://github.com/ultralytics/ultralytics

30/35 meters to have the object captured in the minimum portion of the image needed to be correctly detected by the network.

- The IP performance remains high until the Gaussian noise added to the image is below 12/15 dB. Part of the testing plan will require the analysis of the GNC algorithm for different noise levels.
- Since the camera FOV is conical, for the same distance to the object, the IP algorithm works better when the target is centred in the image because it would depicted with more pixels. Therefore, similar to the guidance and control agent, target pointing is required.

Input	TANGO image based on the Chameleon 3 camera model.
Output	Relative chaser-target position and target orientation in camera reference frame (which coincides with the chaser body frame).

Table 5.2: Input/Output of IP block.

5.1.3 Navigation Filter

The navigation filter, as introduced, is a linear filter for relative translation estimation based on [96], and parametrized following the formulation in [97]. For the same kind of scenario a different formulation was developed in [98] to enhance pose estimation for image-based proximity navigation with uncooperative space objects. Nevertheless, this methodology was not adapted to deal with control action affecting the navigation dynamics as happening in this framework.

Input	Relative chaser-target position in chaser LVLH reference frame.
Output	Relative chaser-target position and velocity in chaser LVLH reference frame.

Table 5.3: Input/Output of NAV block.

5.2 AI-based Guidance and Control

In order to tune the Guidance and Control agent to suit best in the new GNC pipeline, a preliminary training and testing phase is necessary. In this section, these two steps are described and analysed.

At first, the training is constructed in order to teach the agent to be compatible with the new and unknown environment; in particular, three main new characteristics have been updated with respect to the old training analysis performed in the previous chapters:

- **Target object**: is TANGO, with the shape and mesh defined in Fig. 3.4.
- **Camera model**: is based on the Chameleon 3 CM3-U3-13Y3C, already exploited in [99], on which the IG model is developed. The Chameleon 3 is characterized by a FoV of 44.54°x44.54°, a size of 1024 x 1024 pixels, and a focal length of 6 mm.
- **Trajectory range**: is reduced to a minimum of 2 m and a maximum of 35 m. This constraint is driven by the IP block performance, whose estimation drastically decreases when overcoming the 30-35 m threshold in a TANGO-based scenario.

The policy and value networks are the RNN nominal ones defined in Table 3.6 and Table 3.7. The PPO algorithm setting is shown in Table 5.4 with the reward defined in Eq. 4.1.

PPO Hyperparameters	
Reward Discount Factor γ	0.99
Terminal Reward Discount Factor λ	0.95
Clipping Factor ϵ	0.1
Entropy Factor s_2	0.02
Optimizer	ADAM
Optimization Step	each 10 episodes
Optimization Batch	32
Epochs	5
Simulation length	18000 episodes (training)
Policy Network	
Initialization	-
Learning Rate	1e-4
Final Activation Function	Sigmoid
Output Distribution	Categorical
Output Action	Sampling (training) argmax (testing)
Value Network	
Initialization	-
Learning Rate	1e-4
Final Activation Function	-

Table 5.4: Set of PPO algorithm's hyperparameters used for training and testing of the TANGO case study.

The main environmental characteristics, as usual, have been summed up in the following box.

TANGO Case

Chaser-Target: rel. velocity fixed, position and attitude random
Orbit features: Sun phase random, true anomaly θ random
Camera FoV: 40°
Integration step: 20 s for max 86400 s
Trajectory range: 2-35 m
Thrust level: 0.001 m/s^2
Map level: 25 photos/face on **TANGO** shaped object

The resulting trends of the training simulation are depicted in Fig. 5.3. It can be observed that the agent is able to achieve a high level both in terms of reward scoring and map level, almost reaching 90%, in line with the training behaviour in the baseline case discussed in Section 4.2. The time metric is referred to as the trajectory time window.

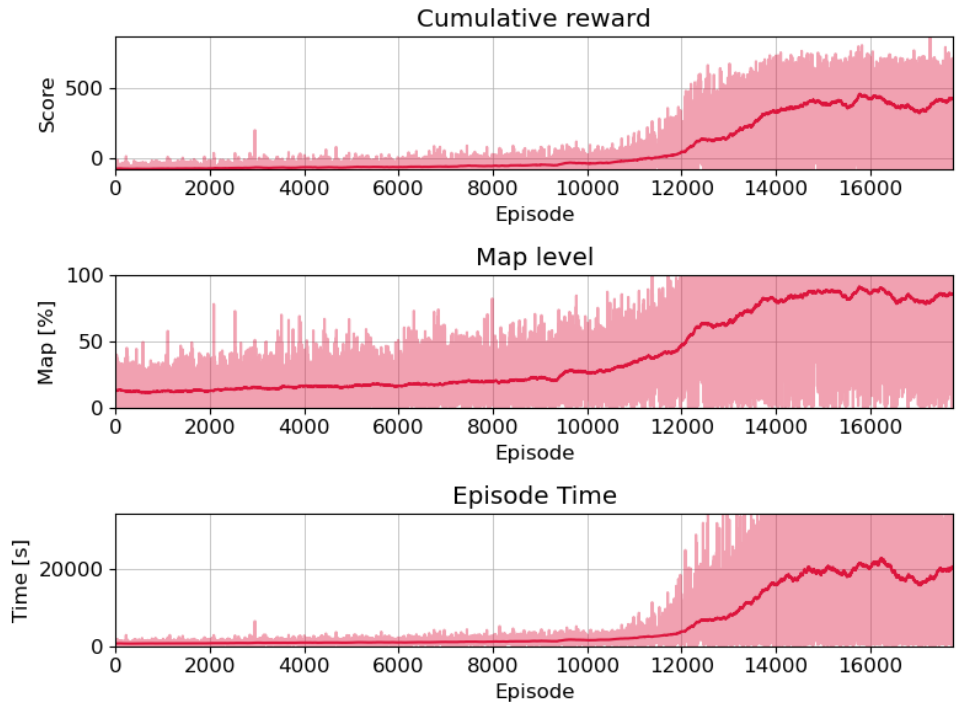


Figure 5.3: TANGO case study training: metrics (reward score, map level, episode time) evolution along the simulation.

In the histogram in Fig. 5.4 the ending condition of each episode of the simulation is plotted; it is observed that the agent escapes the trajectory region almost in the 80% of episodes. The percentage of target collision or complete map reconstruction are around 5.5% and 13.8% respectively. These results

are, of course, affected by the evolution of the agent's experience during the training, and, therefore, must be monitored during the testing analysis.

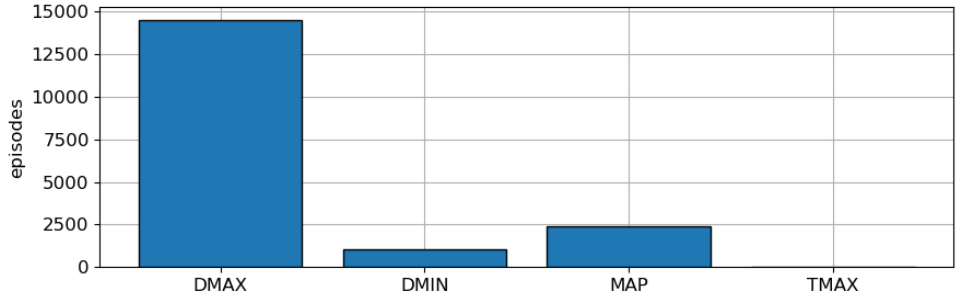


Figure 5.4: TANGO case study training: ending conditions statistics.

Once the training phase is concluded, the outcoming policy network has been tested in order to understand the effective performance in nominal conditions: same random initial conditions, acceleration size, trajectory limit range and completely known input state.

At first, the test is performed still sampling the action (`.sampling()`) from the probability distribution in output from the policy network. In this way, some level of randomness is still present in the simulation in order to keep exploring different actions during testing and detect the possible failure of the agent analysing the simulation's parameters, such as initial or final conditions or number of actions. The average results of this kind of testing are outlined in Table 5.5 and Fig. 5.5.

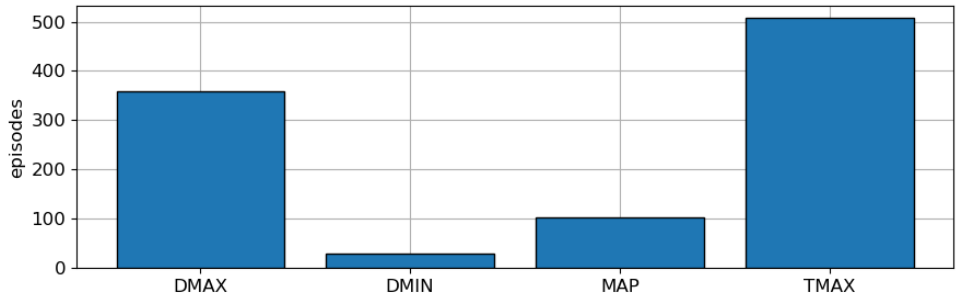


Figure 5.5: TANGO case study testing `.sampling()`: ending conditions statistics.

It is clear that the distribution of ending conditions significantly changes with respect to the training phase. This happens because the agent is fixed and his behaviour does not change during the simulation, which means that now he is capable of maintaining the trajectory within the pre-defined space region, as

evidenced by the increased number of *TMAX* cases with respect to *DMAX* ones. Considering the simulation metrics, i.e. average score, map, time and thrusting level, the obtained results are in line with the expectations. Indeed, bearing in mind that the policy network samples the action to take, the average values are lower than the final performance achieved during the training. In particular, the average thrusting level, that is the percentage of thrust actions taken on the total amount of the action steps in the episode time window, is very high. This is a direct consequence of the sampling action: in the `.argmax()` case, this will strongly differ.

Average Score	Average Map	Average Time	Average Thrusting
142.2	71.7 %	17.1 h	98 %

Table 5.5: Average testing results with `.sampling()` action.

In Fig. 5.6, and Fig. 5.7 the ending conditions of all the tested episodes are investigated with respect to the initial position, and final velocity respectively. From the analysis of the data in Fig. 5.6, it appears that there is no relation between the ending condition of each episode and their respective initial conditions, at least in terms of chaser-target distance.

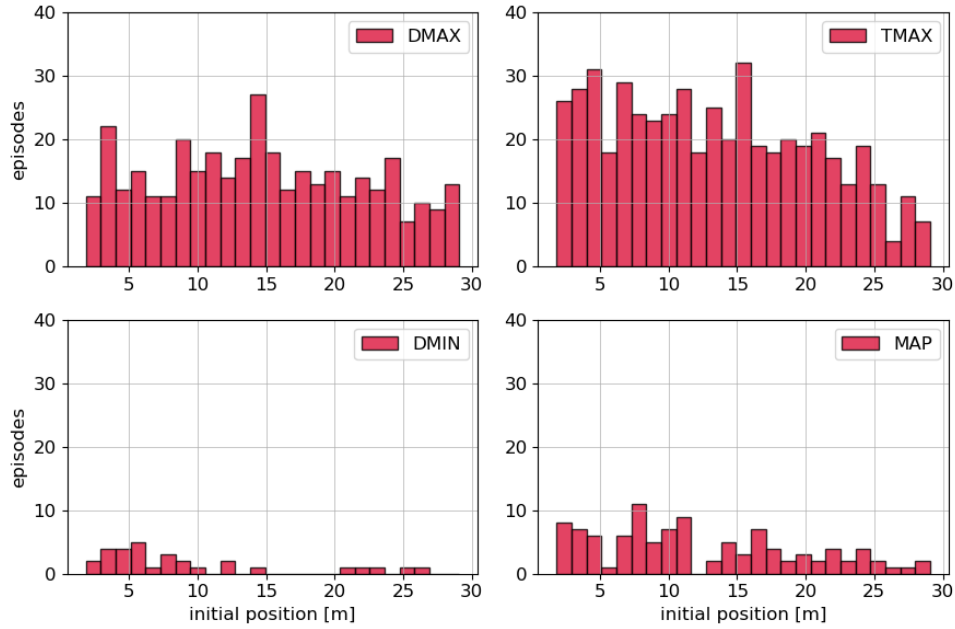


Figure 5.6: TANGO case study testing: ending conditions vs. initial chaser-target relative position.

Almost the same unconnected behaviour can be appreciated in the relation between the *done* conditions and the final relative chaser-target velocity depicted in Fig. 5.7. Only one small characteristic can be observed: if the final relative velocity is higher than 0.1 m/s, it is more likely that the episode terminated overcoming the trajectory upper limit. Nevertheless, due to the very low number of episodes with a final velocity higher than 0.1 m/s, no major assumption can be made, and this particular behaviour is identified as strictly related to the chaser-target intrinsic dynamics rather than the particular agent's failures.

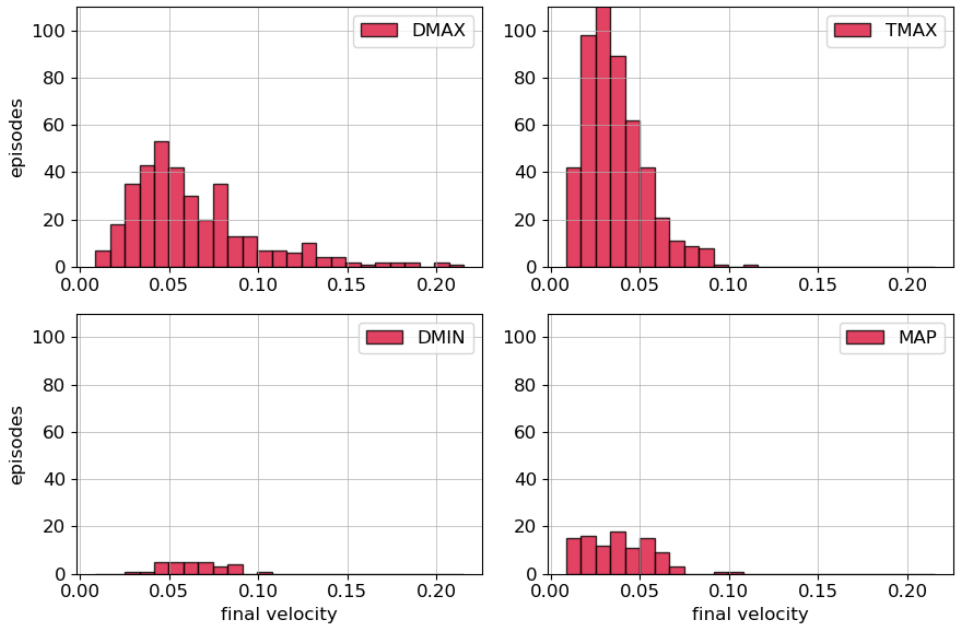
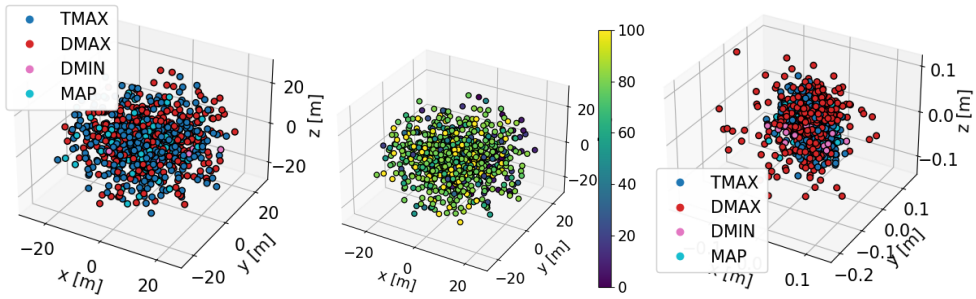


Figure 5.7: TANGO case study testing: ending conditions vs. final velocity.



(a) Initial position vs. *Done* **(b)** Initial position vs. Map **(c)** Final velocity vs. *Done*

Figure 5.8: TANGO case study testing: ending conditions and map analysis.

Also, the analysis of the ending condition and map levels in the 3D space for initial position and final velocity displayed Fig. 5.8 does not bring to any macroscopical incorrect agent behaviour.

In Fig. 5.9 and Table 5.6, the results gathered by the testing simulation in which the policy network selects the action with the maximum probability value through `.argmax()`. As expected the number of episodes that terminate with the *MAP* condition is over 65% of the total; moreover, the average metrics (score, map level and time) perfectly reflect the performance achieved during the training phase shown in Fig. 5.3. In particular, is interesting to observe how both the average time and thrusting percentage are much lower than in Table 5.5: the first, equal to 6.6h, is due to the fact that a lot of episodes stop way before the maximum time window limit since they reach the overall map. The second means that the agent has learned how to properly act without over-controlling the chaser.

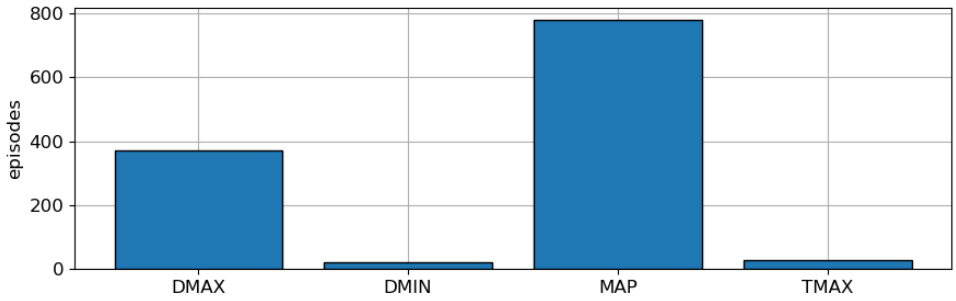


Figure 5.9: TANGO case study testing `.argmax()`: ending conditions statistics.

Average Score	Average Map	Average Time	Average Thrusting
406.4	86.6 %	6.6 h	22.6 %

Table 5.6: Average testing results with `.argmax()` action.

As before, any strong connections in the agent's behaviour have been highlighted between the ending conditions and the chaser-target relative initial position and final velocity (to avoid unnecessary charts), as shown in Fig. 5.10, and with the number of actions in Fig. 5.11. It is interesting to notice that the agent overcontrols the satellite in some cases when the episode concludes overcoming the trajectory limits (*DMIN* and *DMAX*). This may be caused by two opposite reasons: on the one hand, the spacecraft may experience new environment situations which it is unable to face and therefore, overcontrols the satellite, on the other hand, instead, it tries to recover and therefore, over controls the chaser.

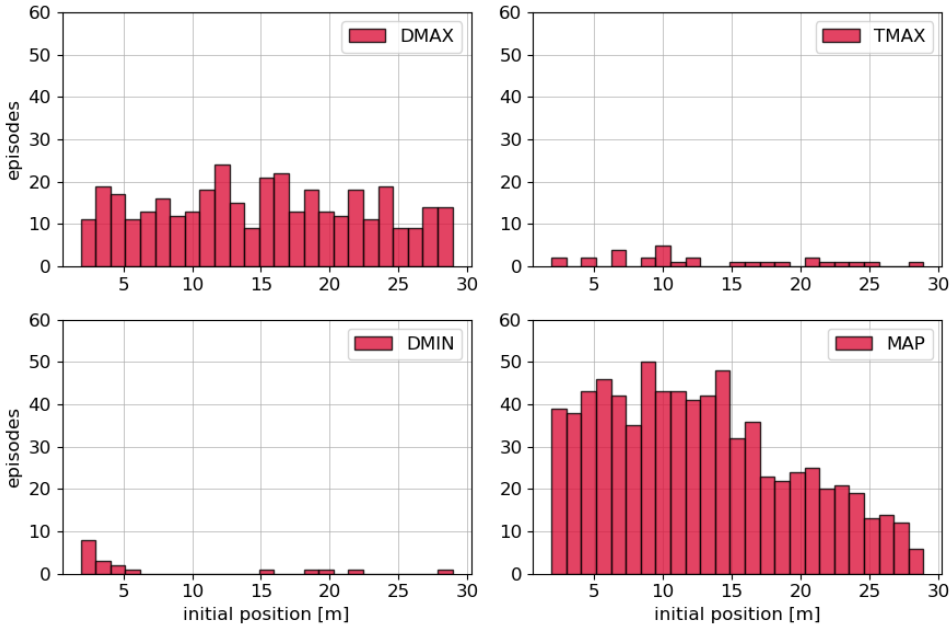


Figure 5.10: TANGO case study testing (`.argmax()`): ending conditions vs. initial chaser-target relative position.

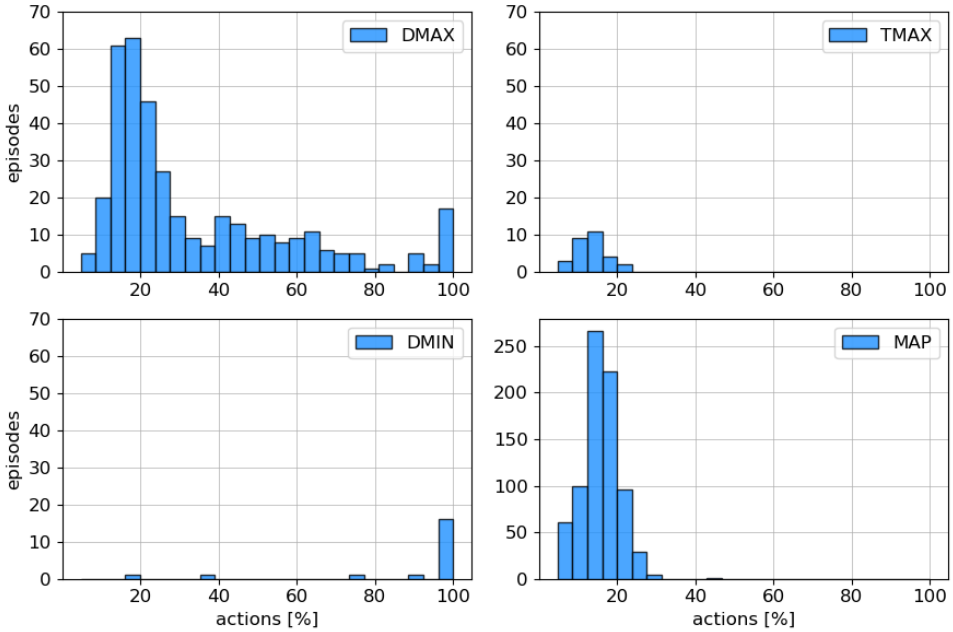
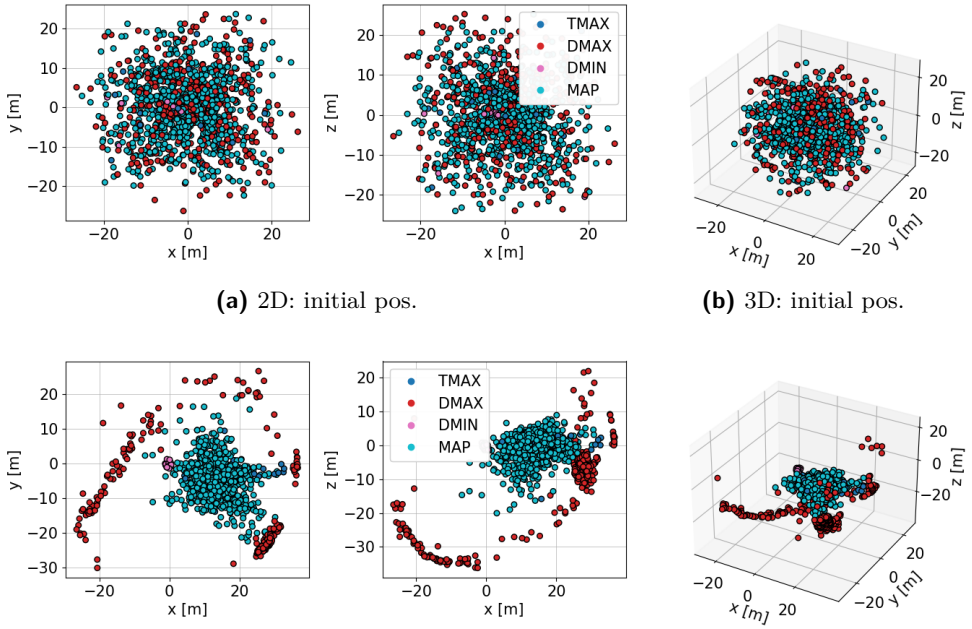


Figure 5.11: TANGO case study testing (`.argmax()`): ending conditions vs. final velocity.

In order to analyse more in-depth this particular behaviour, also the relation in 3D space has been studied, as shown in Fig. 5.12. No connection seems to be present between the initial relative position and ending conditions as observable in (a) and (b). Instead, it is interesting to notice the agent's behaviour with respect to the final relative position depicted in (c) and (d): indeed, the agent seems to have learnt a particular set of paths through which the mission can be accomplished maintaining its relative position in the [+x, -y] quadrant. This particular feature, which will be reconfirmed again also in the next Section 5.3, is mostly related to the sub-optimal policy that the agent has acknowledged in the training phase.



(a) 2D: initial pos. **(b)** 3D: initial pos.

(c) 2D: final pos. **(d)** 3D: final pos.

Figure 5.12: TANGO case study testing (`.argmax()`): ending conditions vs. initial/final position in 3D.

Once clarified that no particular failures characterize the agent behaviour in a specific region (intended in a broad sense) of the environment, it is therefore possible to perform the sensitivity analysis on the trained policy.

5.2.1 Sensitivity analysis

Several sensitivity analyses have been performed on the trained network in order to assess the robustness of the agent that needs to interact with a slightly different environment and image processing and navigation blocks, not always

coherent with the characteristics on which it has been trained. In particular, the testing cases are reported in the following list:

- Random initial conditions with higher discretization range.
- Different acceleration sizes.
- Different integration steps.
- Noisy input considering the error level of the IP algorithm.

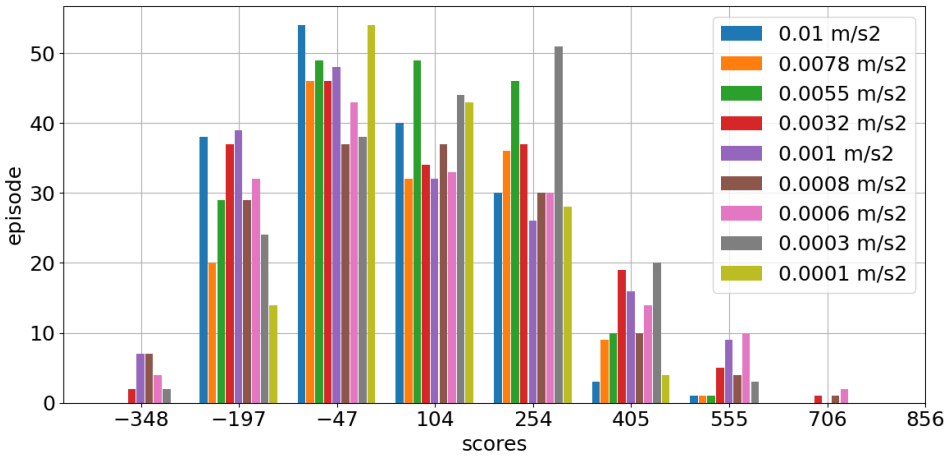
Random Initial Conditions. At first, the agent is tested increasing the randomness level of the state's initial conditions in terms of discretization. The discretization affects the randomness level of several variables such as chaser-target relative position and orientation, Sun phase and orbital true anomaly. This means that the range limit remains unchanged while the granularity of the input is increased in order to span a more complete portion of space. In Table 5.7 the results of the analysis have been reported: the agent is completely robust to this increased complexity. Indeed, the evaluation metrics (score, map and time) maintain the same level.

	Discretization		
	Nominal	100%	200%
score	406.4	390.5	404.5
map	86.6 %	85.9 %	87.1 %
time	6.6 h	6.2 h	6.4 h
thrust	22.6 %	22.6 %	21.8 %

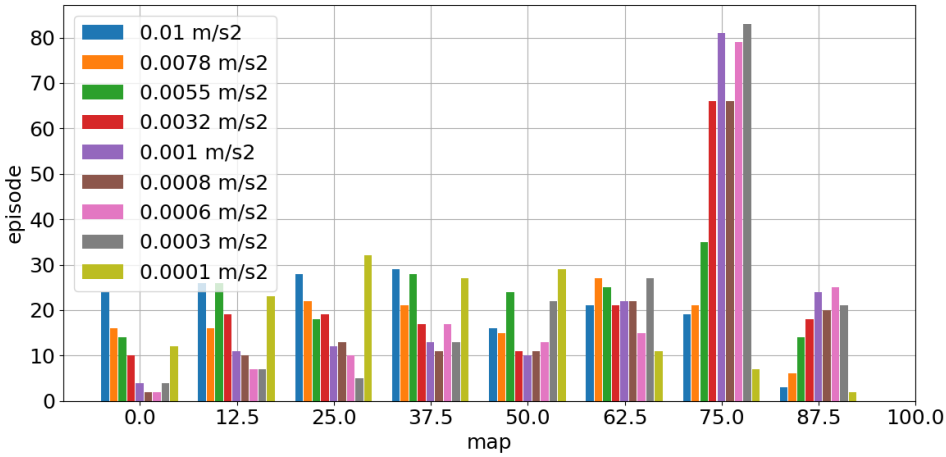
Table 5.7: Sensitivity analysis on random initial conditions: discretization and simulation metrics.

Acceleration Size. One of the main assumptions made in the environment definition explained in Section 4.2 and Section 5.1 is the model of the action space of the agent. In this particular case, a discrete action space is defined with a level of acceleration fixed at 0.001 m/s^2. This sensitivity analysis changes the level of acceleration at each episode, randomly selecting it among a set of possibilities defined between one order of magnitude above and below the nominal one. Therefore, the pool of choices is limited between 0.01 m/s^2 and 0.0001 m/s^2. The plots in Fig. 5.13 summarise the behavioural trends of the agent with respect to these different sizes focusing on the episodes' final scores and map levels. It can be noticed that the central block of options, the one nearer to the nominal one, behaves similarly and obtains metrics at a level close to the nominal one. On the contrary, the limit values are the ones that behave worst. This occurs especially for the highest values, which clearly make

the chaser less controllable since the action would affect more strongly the relative dynamics, as happens for the lowest value, which is unable to modify the dynamics significantly.



(a) Acceleration vs. Score



(b) Acceleration vs. Map

Figure 5.13: Sensitivity analysis on different ranges of acceleration size: comparison to simulation scores and map levels.

Integration Step. The nominal integration step on which the agent has been trained is equal to 20s. This sensitivity analysis on these parameters is important to understand how the agent behaves when the step is smaller or greater, contingency that may occur due to temporary slowdown of the image processing or navigation steps, or due to different on-board software frequency

requirements, if lower or higher. As observable in Table 5.8, the metrics related to the lower integration step of 10s are even higher than the nominal case ones. This means that the agent behaves well or better when he has more control of the spacecraft, while the performance starts decreasing when the control windows are wider.

	Integration step			
	10 s	20 s (Nom)	30 s	40 s
score	**459.8**	336.9	255.9	134.0
map	**75.6** %	73.7 %	66.7 %	50.7 %
time	**12.8** h	12.1 h	9.3 h	5.2 h

Table 5.8: Sensitivity analysis on integration steps: steps levels and simulation metrics.

Noisy Input. Concerning the sensitivity of the noisy input, the analysis is similar to the one extensively performed in Section 4.3, even if more shaped on the GNC pipeline block characteristics. Therefore, instead of considering the noise of the input as suggested by [73], as done in the previous chapter, here the performance level of the IP and NAV blocks is considered.

Noise	**Input**	**Level**	**Metrics**	**Avg. Val.**
Nom	rel. pos.	3 %	score	323.5
	rel. vel.	3 %	map	85.2 %
	ang. pos.	8 %	time	5.1 h
	ang. vel.	8 %	thrust	28.4 %
Mid	rel. pos.	5 %	score	268.2
	rel. vel.	5 %	map	81.23 %
	ang. pos.	10 %	time	4.7 h
	ang. vel.	10 %	thrust	36.5 %
High	rel. pos.	5 %	score	205.5
	rel. vel.	5 %	map	76.8 %
	ang. pos.	20 %	time	5.0 h
	ang. vel.	20 %	thrust	37.2 %

Table 5.9: Sensitivity analysis on noisy input: noise levels and simulation metrics.

In particular, the error levels proven in [79] are around 3% for the chaser-target relative position and velocity and around 8-10% for the relative attitude. The

results outcoming from the test are reported in Table 5.9. As expected, the performance decreases with the increase of the input errors.

Finally, after having analysed in depth the robustness and performance of the trained agent, it is time to test this DRL based guidance and control policy as one of the blocks composing the overall GNC pipeline.

5.3 TANGO fly-around case study

This section describes the testing campaign done to analyse the performance of a complete AI and image-based GNC pipeline. The algorithm schematic and architecture have already been presented in Fig. 5.1. Therefore, starting from the different features of each of the blocks involved in the algorithm, a testing workflow has been followed in order to increase the complexity of the simulations step by step, without rushing directly to the most general case. The adopted procedure is summed up in Table 5.10. It can be noticed how a new level of complexity has been added at each new simulation, starting from the noise on the guidance and control block input, passing through increasing the IP block noise, and ending with the addition of the attitude control for target pointing.

ID	In. Cond.	Range	GC input	IP noise
1	GC	1.5-35 m	rv - NAV q - DYN	3 dB
2	GC	1.5-35 m	rv - NAV q - DYN + noise	3 dB
3	GC	1.5-35 m	rv - NAV q - DYN + noise	6 dB
4	GC	1.5-35 m	rv - NAV q - DYN + noise	12 dB
5	GC	1.5-35 m	rv - NAV q - IP + noise	12 dB
6	GC + att. cont.	1.5-35 m	rv - NAV q - IP + noise	12 dB

Table 5.10: Planning of the testing simulation to assess the feasibility of the AI and Image-based GNC pipeline.

5.3.1 Testing Analysis

Before entering into the details of the results, it is important to recap some of the main assumptions that have been considered in the overall pipeline due to input constraints of the different blocks:

- The **Earth** is never in the **background** of the image for two main reasons: the guidance reward model was not built considering Earth illumination and the IG block would have been too slow in rendering the images to maintain a feasible simulation time.
- The guidance and control agent is trained on a **fixed target pointing** scenario, which at first, does not require any attitude controller.
- The **trajectory** is **limited** to remain between 1.5-35 m from the target in order to avoid collisions or losing the IP estimation.

These assumptions lead to some tests on the GNC pipeline, focusing on the guidance and control (GC) block. The first concerns the input of the chaser-target relative orientation, which could have been derived or adding a proper level of noise to the real quaternion value (computed by the dynamics block) or taking it from the IP estimation. The second is related to attitude control, which has been added as the last step of the plan, removing the assumption of fixed target pointing. It is worth explaining these two cases more in detail.

GC attitude input. As just introduced, the attitude information needed by the GC agent can be derived directly by the DYN or by the IP blocks. In the first case, the estimation is less realistic because only noise on the real quaternion is being considered. Here, the noise is added following the model used in [100]. In the second case, the input is more proper since it comes from the IP estimation. Nevertheless, also for this formulation, another assumption must be considered: indeed the IP block outputs the chaser-target relative orientation in the chaser camera reference frame, while the GC agent needs it in the LVLH one. Therefore, in order to rotate the quaternion from the body to LVLH frame, the knowledge of the absolute position of the target object must be assumed, which can be still computed knowing the estimation of the relative position and the chaser absolute one.

Chaser attitude control. Once the fixed target pointing assumption has been removed, a PD control has been activated in the dynamics block, computing the target pointing with the quaternion estimator algorithm (QUEST) [101]. The proportional and derivative gains are not fine-tuned on the chaser in order to leave a small level of uncertainty on the attitude control to analyse the robustness of the GC. Moreover, the coupling between the trajectory and attitude control is not taken into account on the one hand not to increase too

much the complexity, on the other because the GC agent is not trained for this kind of scenario. Nevertheless, this test will explore a lot of states never experienced by the agent, it will be able to act properly anyway.

The results of the test analysis are shown in the next pages. In Fig. 5.15, Fig. 5.16 and Fig. 5.19 one episode of ID:5 test simulation is outlined as an example. In Fig. 5.17, Fig. 5.18 and Fig. 5.20, instead, one episode of ID:6 test.

- In Fig. 5.15 and Fig. 5.17, the relative chaser-target position and velocity (highlighting when the agent takes active control on the chaser), the simulation metrics (score, map level and thrust percentage), and the trajectory around the object are plotted. From the graphs, it is easy to notice how the agent tries to keep the spacecraft at an average relative position and velocity value, actively controlling it only these two parameters start to drift away from this value.
- In Fig. 5.16 and Fig. 5.18, instead, the errors between the real dynamics and the estimation of the IP and NAV blocks are shown. As predictable, the error of the IP estimation increases together with the relative position, still remaining very precise, as well as the NAV estimation, which never exceeds 2/3% or 0.5 m at maximum distances. The same happens for the relative velocity estimation, which never overcomes 0.01 m/s on the maximum value of 0.05 m/s. In both cases, the x and y axes have higher errors than the z axes. This is related to the dynamic model exploited in the environment integration.
- At last, in Fig. 5.19 and Fig. 5.20, the chaser-target attitude error is shown, between the real value and the one derived by the IP estimation. On average the error is around 15°-18°: it is such high because of the uncertainty of the IP estimation (3°-5° at maximum), a further noise has been added to the chaser absolute attitude value knowledge necessary to rotate the quaternion into the GC input reference frame, as stated in the section before.

Considering the performance of the trained agent analysed in Section 5.2, it is reasonable to expect that at least 4 episodes on 5 would succeed (reaching the 100% of map or anyway remaining inside the trajectory limit ranges) in a single simulation.

Each of the tests listed in Table 5.10 run on 5 episodes simulation length due to heavy computational time, with the usual stopping conditions: trajectory limits (range between 1.5-35 m), time window (set to maximum 4 hours of trajectory length). and map level (if reaches 100%). The results are reported in Table 5.11 and show that the agent performance remains more or less constant during all the tests, proving the great robustness of the trained model. Even if the sample of episodes is very short, just up to 5 for each simulation, the fact that the

ID	Scr.	Map	Time	Thr.	Ending Conditions			
					DMAX	*DMIN*	*MAP*	*TIME*
1	372.2	81.7 %	2.7 h	48.3 %	1	-	2	2
2	341.2	81.3 %	2.6 h	40.7 %	1	-	3	1
3	437.6	86.8 %	3.6 h	33.1 %	1	-	1	3
4	414.4	93.7 %	3.1 h	34.5 %	1	-	3	1
5	378.2	89.5 %	2.7 h	28.6 %	1	-	3	1
6	505.8	84.3 %	3.2 h	29.4 %	2	-	-	3

Table 5.11: GNC pipeline results.

metrics levels are almost the same in all the tests supports the findings. The fact that in some simulations the average metric is lower or higher with respect to the others is then connected to the single episode and not to the average performance. Indeed, it is easy to notice how, on average, the evaluation metrics respect the standards of the nominal ones. Considering the ending conditions, it is interesting to notice that, differently from the previous ones, only the ID:6 never gets the maximum map level. This may be caused by the additional attitude control aspect in the environment, which, not guaranteeing anymore a fixed target pointing, may lower the agent's performance.

In the end, considering, the overall results, the DRL trained GC agent seems to be perfectly capable of interacting with realistic IP and NAV blocks in a GNC closed-loop pipeline.

One last analysis, shown in Fig. 5.14 has been carried out on the comparison of the agent behaviour when acting in the GNC environment or in the training environment defined in Section 5.2, considering fixed target pointing. In blue is depicted the relative position and velocity when the agent acts in the GNC algorithm, thus exploiting the input of the IP and NAV blocks. On the contrary, in orange, is the trajectory's variable followed by the chaser starting from the same initial conditions, but taking as input the real values as happening during the training phase. It can be noticed that, even if they clearly fly-around the target with two different trajectories, they still accomplish the same objective. This means that for the agent knowing exactly its current state is not strictly important as long as the input represents a reasonable state condition. This very powerful feature is a direct consequence of the exploitation of a guidance and control policy approximator trained through deep reinforcement learning. For sake of clarity, the yellow line in Fig. 5.14 depicts the *free-dynamics* state, demonstrating again how an uncontrolled trajectory is totally unable to even approach an intermediate result.

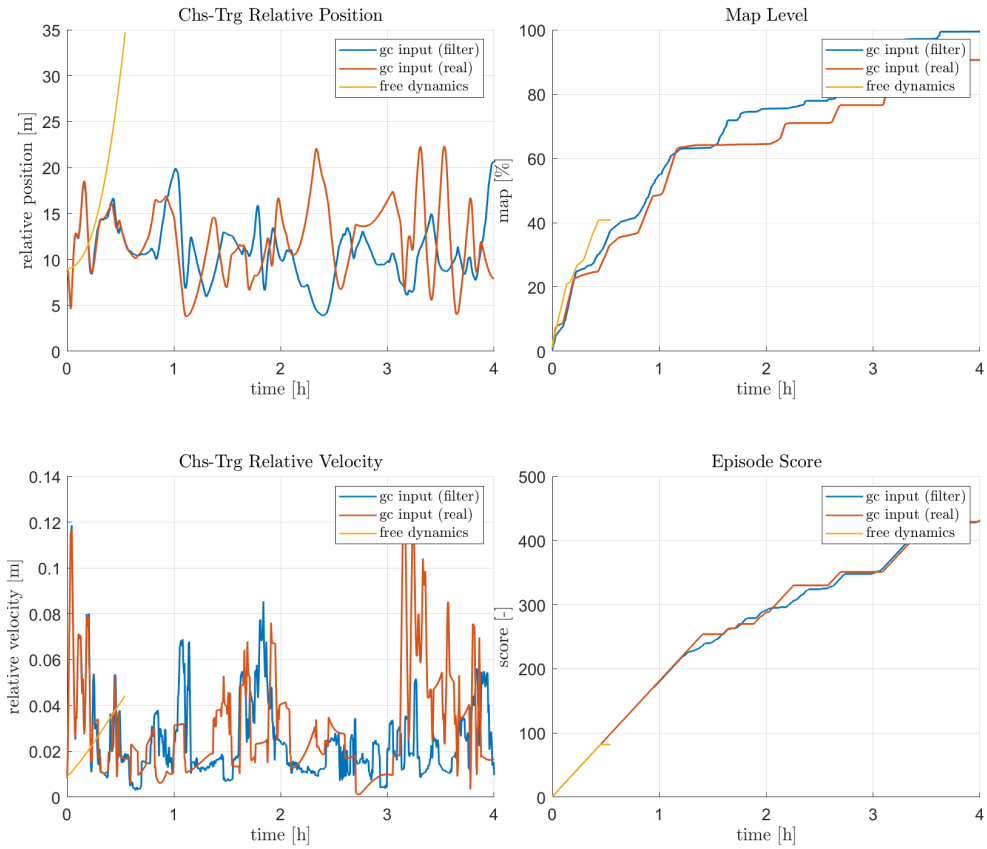


Figure 5.14: Comparison of GC agent generated trajectory within the GNC algorithm and in the training environment defined in Section 5.2.

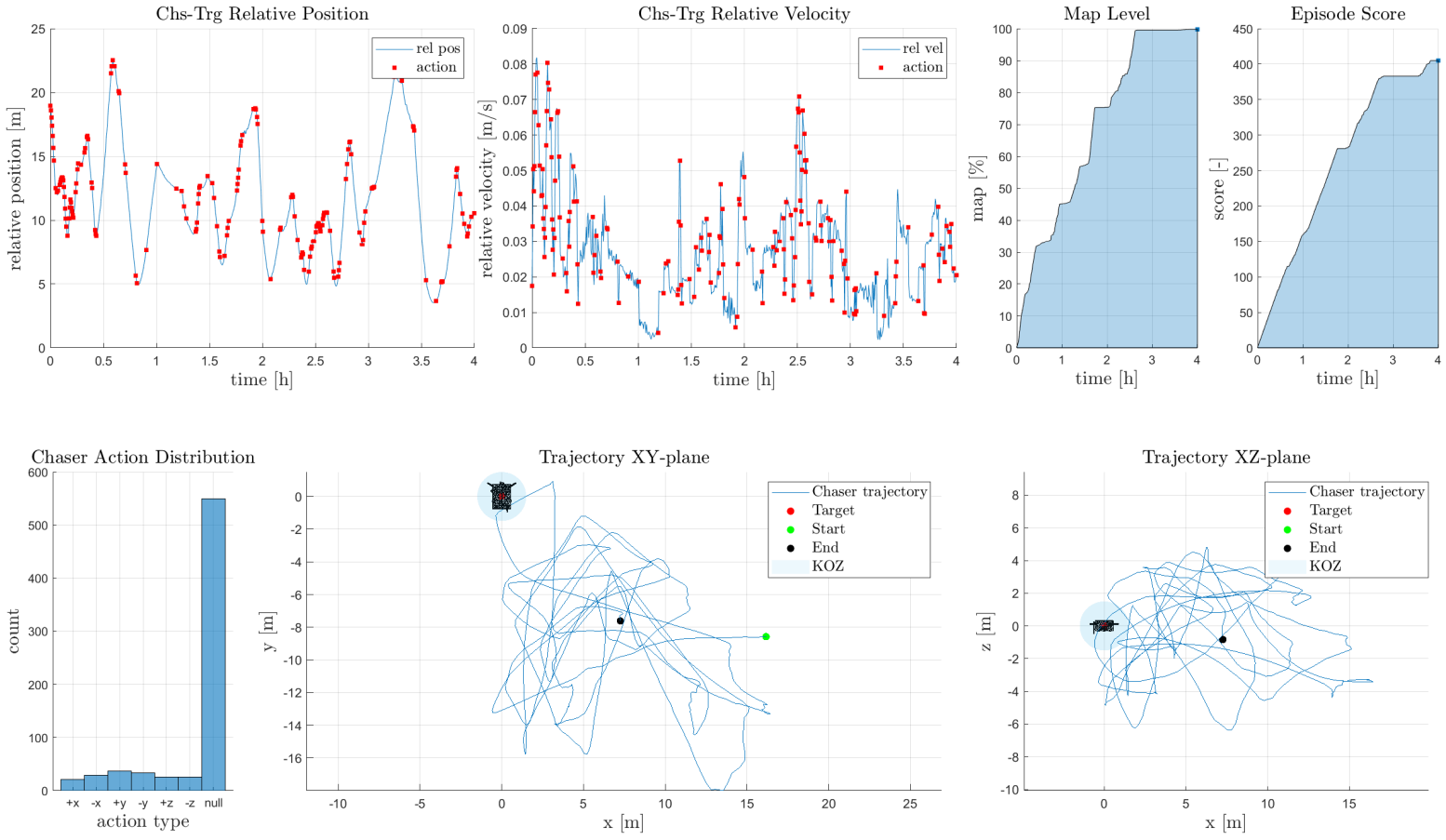


Figure 5.15: Example of Chaser-Target trajectory and evaluation metrics in the *fixed* target pointing case.

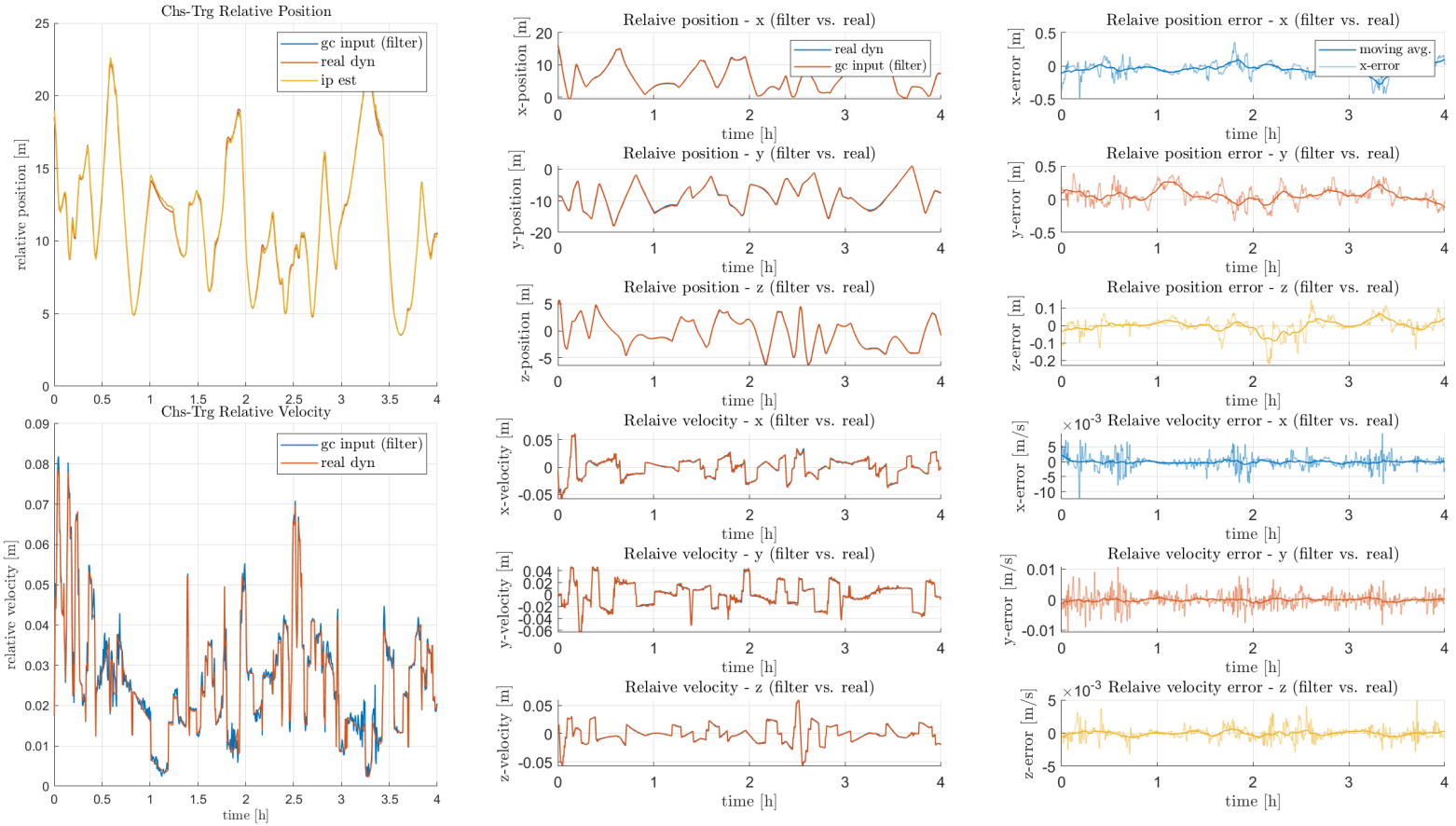


Figure 5.16: Chaser-Target relative position and velocity errors in the *fixed* target pointing case.

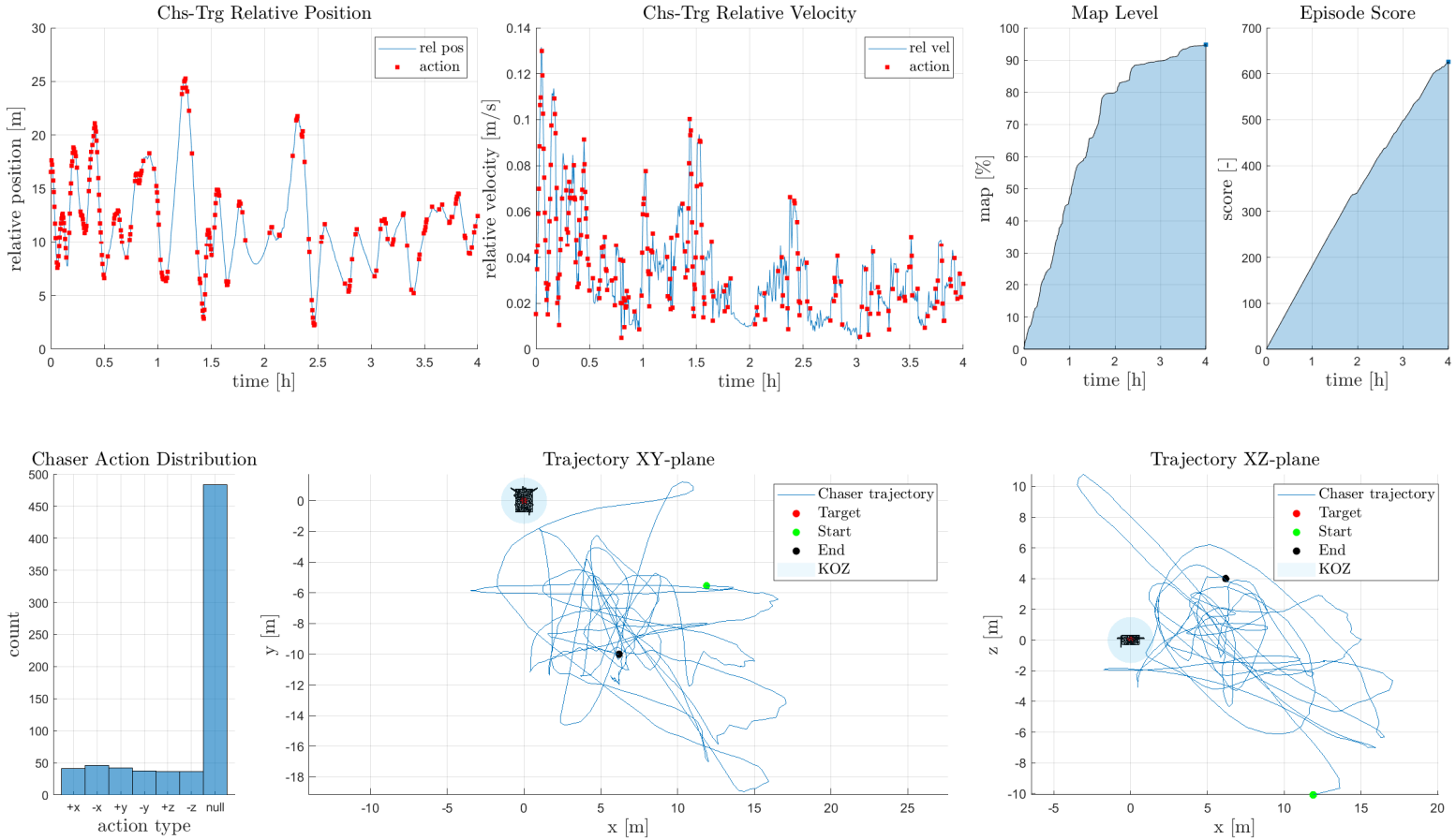


Figure 5.17: Example of Chaser-Target trajectory and evaluation metrics in the *controlled* target pointing case.

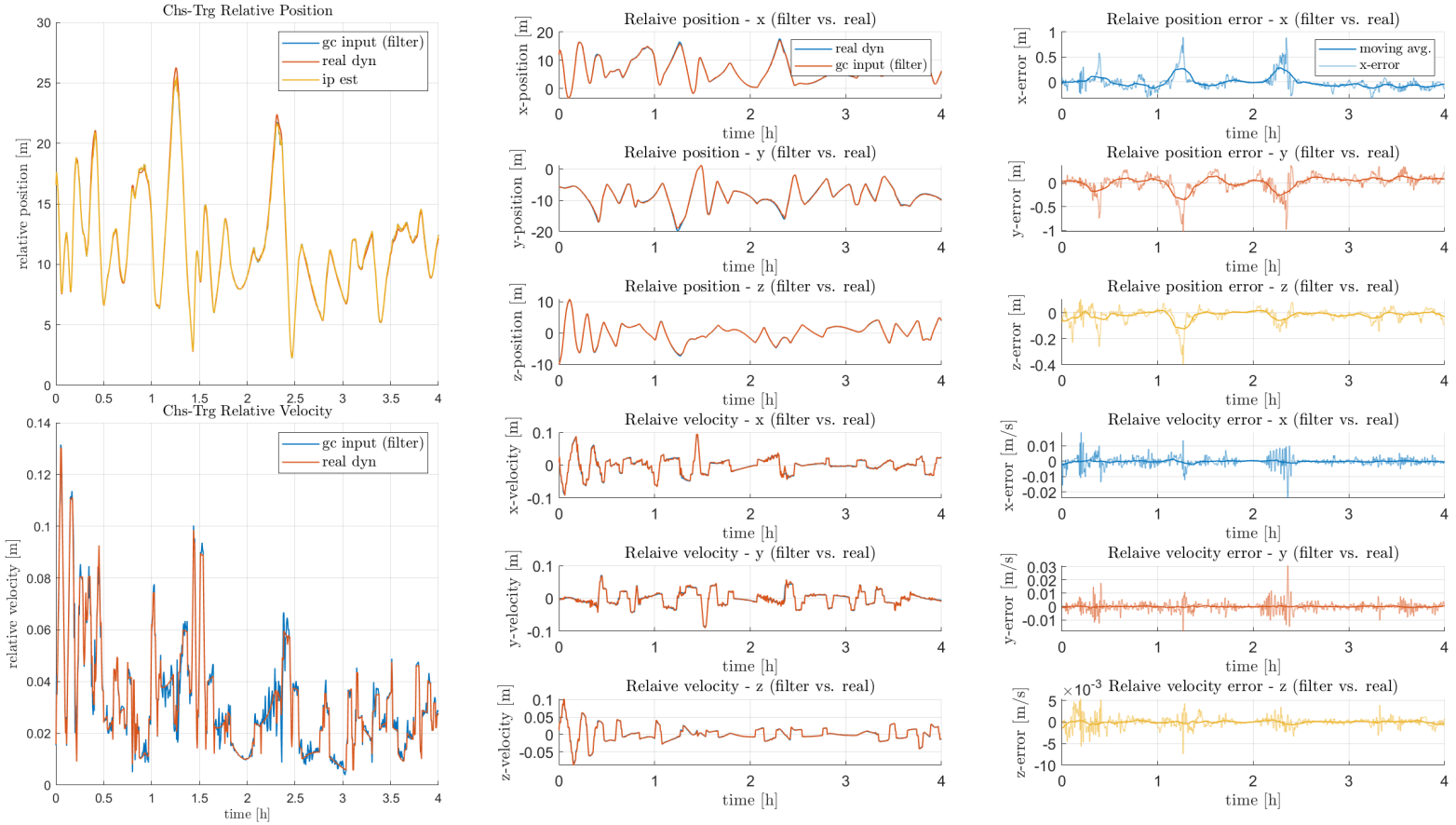


Figure 5.18: Chaser-Target relative position and velocity errors in the *controlled* target pointing case.

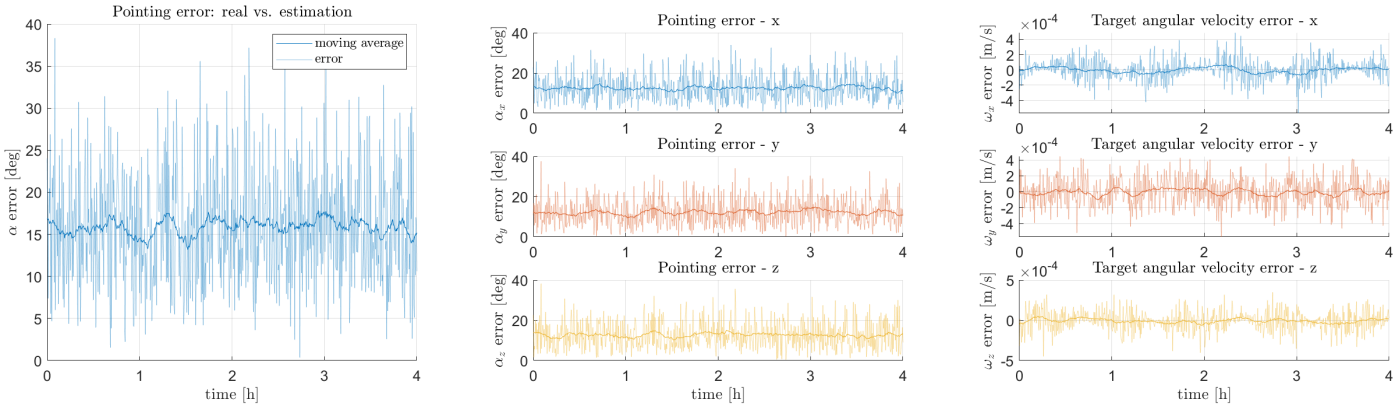


Figure 5.19: Chaser-Target relative attitude errors in the *fixed* target pointing case.

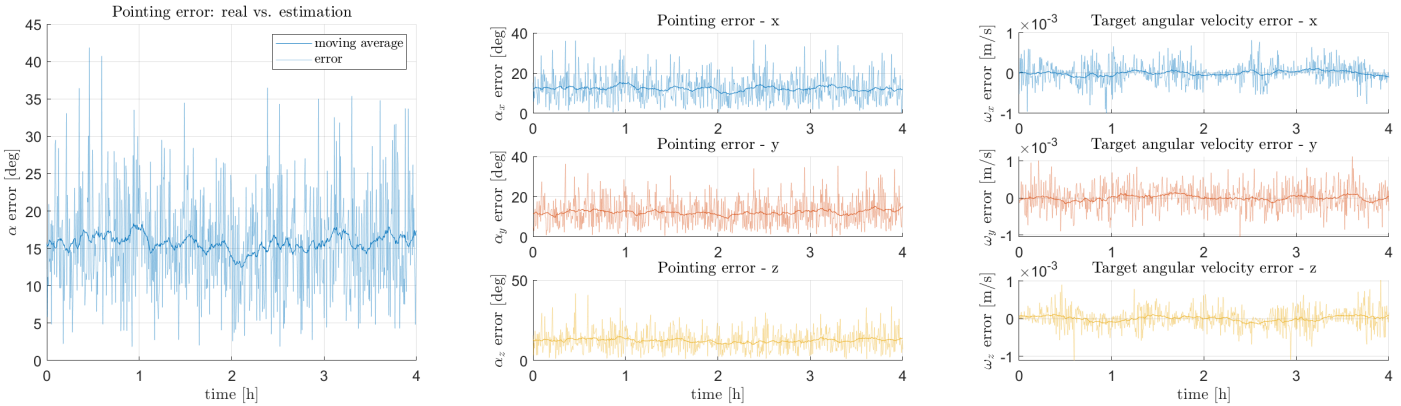


Figure 5.20: Chaser-Target relative attitude errors in the *controlled* target pointing case.

5.4 Closing Remarks

In this chapter, the proposed GNC algorithm architecture has been presented in all of its components. In particular, the guidance and control block is treated in depth, at first, analysing the training problem shaped for the TANGO environment, up to entering into the details of the extensive testing and sensitivity analysis campaign. Once the agent's level of robustness and flexibility has been assessed, the developed block was inserted and interfaced into the image-based navigation tool. As shown in the previous pages, the trained agent exhibits an almost stable and unaltered level of performance throughout all the steps of the testing simulations summarized in Table 5.10, although some of those were quite distant with respect to the environmental model in which the agent itself has been trained, i.e. controlled chaser's attitude dynamics. The policy learned seems to be robust to all the uncertainties and model differences introduced in the new GNC framework: this is demonstrated by the trajectory path that the agent follows, or at least tries to follow, at each simulated episode. Moreover, the intrinsic estimation errors generated by the image processing and navigation filter do not affect the performances even if they are greater or comparable to more classical navigation techniques. In conclusion, even if some points still remain open, e.g. coupling compatibility between trajectory and attitude control, absolute attitude knowledge assumption, etc., the main objective of this analysis, as defined in Section 1.1, may be considered achieved with good success. Of course, these results have strengthened the awareness of having a powerful tool, that, at this point, feels the need of being tested also in a realistic or hardware-in-the-loop environment. More on this future development will be discussed in Section 7.1.

CHAPTER 6

Assessment of DRL for Guidance in Space Applications

> And I knew exactly what to do. But in a much more real sense, I had no idea what to do.
>
> — Michael Scott, *The Office*

In this chapter, the methodology developed in the previous ones will be summed up and tested in different space applications characterized by the same level of complexity in terms of environment conditions, state-action space and reward modelling. These new applications have been investigated through the overall development of the thesis, as depicted in Fig. 1.1. These analyses helped to characterize and assess the feasibility of the approach focused on the training and testing pipeline definition. In Section 6.1, the methodology is summarized and the approach is explained. In Section 6.2, three different applications have been reported and investigated at three different levels of process development.

6.1 Overview of the Methodology

The proposed methodology has been applied to the development of the PPO-based agent in Chapter 4; it is based on three main analyses that involve:

- **Model selection**: in terms of network architecture and state-action space definition.
- **Reward modelling**: which includes the selection of the complexity of the mission's objectives.
- **Initial conditions robustness**: where all the parameters involved in the random selection and their relative randomness level are defined or designed.

The adopted workflow based on these three aspects is schematized in Fig. 6.1. This methodology is proposed as a sort of *vademecum* or guidelines to follow for the implementation and testing of a robust DRL-based path or strategy planning agent. Indeed, all three scenarios that will be analysed in the next section make use of and benefit from this workflow philosophy, whether fully or partially.

From a superficial layer point of view, the rationale behind this structure comes from a very obvious functionality: testing the approach on a simpler and easier problem helps in understanding the initial performances of the approach and finding out which are the most important points to investigate more in-depth. As shown in Fig. 6.1, the proposed workflow differentiates between two kinds of scenario definitions, characterized by two different levels of complexity. These levels can be shaped both by the definition of the model and by that of the reward function. The model depends on many factors and aspects, but, among all, is mainly described by the network architecture and the state-action spaces; on the other side, the reward formulation embeds the mission objectives. Once these two, or more if needed, cases have been selected, the workflow proposes to train and test them with different levels of initial conditions. This difference can be interpreted in distinct ways: for instance, in Section 4.3, it concerns the randomness level of the initial conditions; while, in other cases, may be related to the selection of a simplified or realistic subset of them. In the end, what matters is that in one case, the state space possibilities are reduced with respect to the other. In Fig. 6.1, the distinction between the methodology steps (1 to 4) and the *feedback* steps is highlighted by the different lines' colours: black and red. This wants to underline the adaptability of this *vademecum* which must face the need of tuning the process without precluding any possibilities (in terms of model and reward) in defining them *a priori*. To summarize, taking the scenario presented in Chapter 4 as a reference, all the aforementioned aspects have been analysed: in Section 4.2 the differentiation in ANN architectures and rewards' shaping is treated and investigated. Therefore,

these four coupled models are trained with two different types of random initial conditions following the strategy outlined in the Fig. 6.1. In this way, it was possible to understand how the agent is robust and adaptable. In Section 4.4 the same kind of analysis has been followed to increase again the complexity of the model, in this case in terms of state and action space.

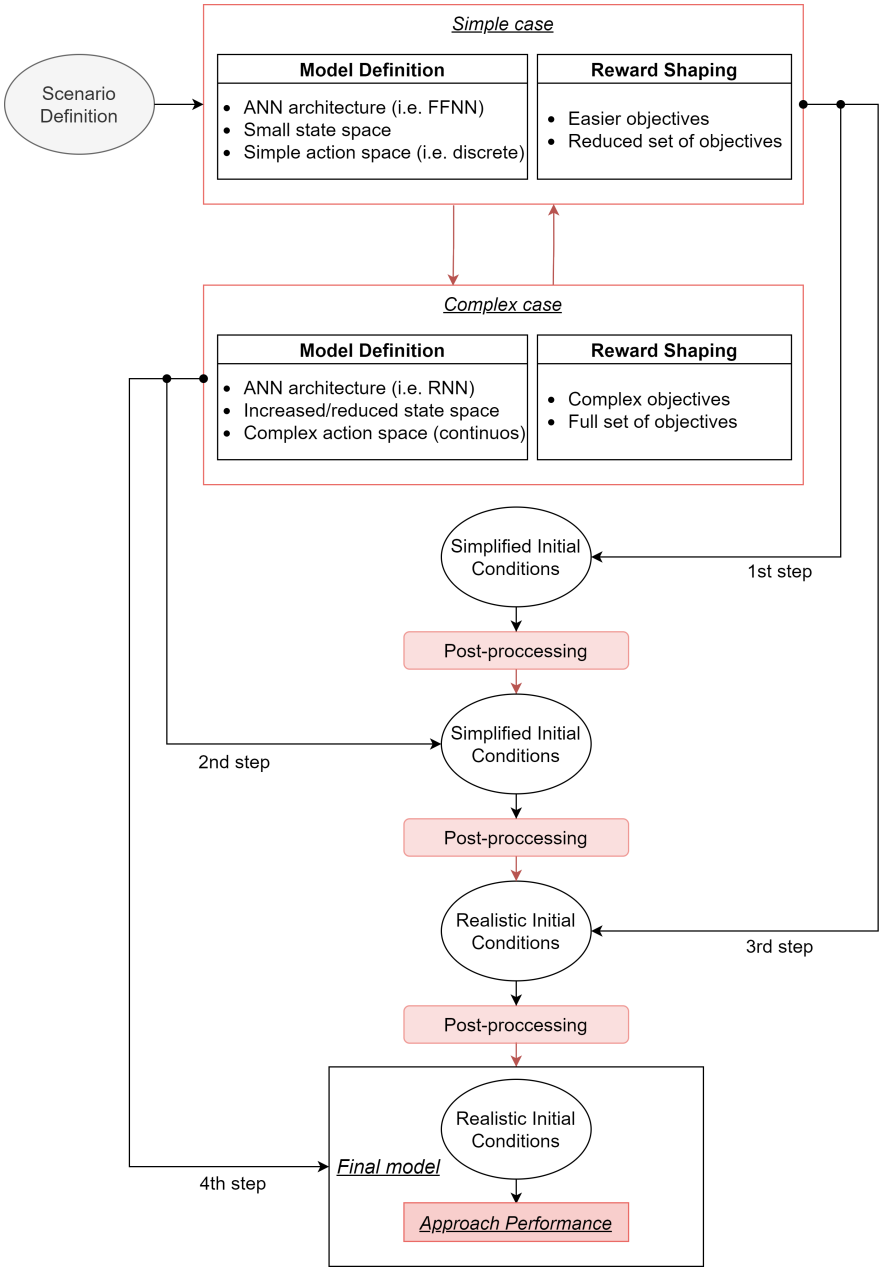


Figure 6.1: Schematization of the proposed workflow methodology.

This kind of philosophy may not be strictly followed but instead should be adaptable case by case. Indeed, as introduced before, the following three examples adopt this proposed philosophy, adjusting it in relation to their desired needs.

6.2 Space Applications Analysis

In the following subsections, the three different applications will be briefly analysed, focusing more on the approach methodology in terms of training and testing design. The first application regards the debris collision avoidance strategy planning of LEO spacecrafts; the second is related to the guidance optimization of the launcher's first stage landing; finally the third concerns again the guidance and control optimization of a robotic arm aimed to capture uncooperative targets.

6.2.1 Debris Collision Avoidance's Strategy Planning

***State-of-the-art* and Motivation.** Nowadays the massive deployment of small satellites and large constellations in particular regions of space, i.e. Low Earth Orbit (LEO), could become a major source of safety problems to space operations, which are already undermined by the great number of debris orbiting the Earth. Each week, hundreds of warnings on possible encounters with other spacecrafts or space objects are detected and issued for lots of LEO satellites. These alerts, named Conjunction Data Messages (CDMs), contain information on the time of closest approach, miss distance, relative position and velocity of the two objects, together with the time of last accepted observation[1]. The risk of collision may decrease as the week goes by, or, on the contrary, may increase enough to take further action, which could involve hours of analysis. The number of CDMs has been rapidly increasing in these years, and the current manual process for collision avoidance and manoeuvrer strategy may, soon, be too slow and time-consuming to be sustainable. For this reason, leading space agencies are investing in the automation of collision avoidance processes. The definition of fast and reliable analytical or semi-analytical approaches suitable for on-board implementation may represent the first step towards the automation of collision avoidance manoeuvre (CAMs) planning. These kinds of approaches have been applied to strategies based on low-thrust manoeuvrers both in terms of energy-optimal control problem [102] or proximal motion equations, taking also into consideration the effects of drag and solar radiation pressure [103]. In [104] a new accurate and relatively simple analytical expression for the computation of the relative dynamics of two colliding objects in the *b-plane* has been proposed; it is based on the characteristics of the orbit of the manoeuvrable spacecraft, the encounter geometry and the impulsive

[1] `https://www.space-track.org/documents/CSM_Guide.pdf`.

manoeuver characterization in terms of magnitude, orientation, and phasing. Moreover, taking advantage of the constantly growing use of machine learning tools, some works have already analysed the potentiality of ML on this collision avoidance problem, even if it remains a relatively unexplored field [105, 106]. One of the most significant analyses is provided in [107], where ML techniques are combined to uncertainty quantification and orbital mechanics calculation based on an available dataset of CAMs, created by simulating a range of scenarios in which optimal manoeuvrers are applied to reduce the collision probability. Here, predictive models are trained to forecast the risk of avoiding new collisions, suggesting alternative manoeuvrers. Moreover, in order to augment the computational speed of the manoeuvrer optimization module by providing accurate estimates with a limited amount of training data, an open-source collision avoidance manoeuvrer tool, namely OpenCAMPT, has been developed [108].

Problem definition and Results. The analysis that will be briefly presented here, wants to investigate the applicability of DRL techniques to the design of sub-optimal long-term impulsive collision avoidance strategies. This work is developed in the framework of [109]. The workflow followed for the training and testing analysis reflects the one proposed in the previous section, as depicted in Fig. 6.2.

- **Environment Complexity.** The scenario environment is based on the synthetic generation of space debris through the ESA's *Meteoroid and Space Debris Terrestrial Environment Reference* (MASTER-8), which considers the actual distribution of the conjunction events along one orbit. The quantities retrieved from MASTER-8 are then used to compute the collision avoidance probability with Chan's analytical expression [106], and the energy-to-mass ratio. The collision avoidance manoeuvrer is based on Bombardelli's model [104].

- **Model Architecture.** The problem can be formulated as a POMDP problem, and, therefore, it can be solved with DRL methodology. In this case, an A2C approach has been followed, defining a continuous state and discrete action spaces. The state model is differentiated into a simpler and a more complex space; both are composed by the location of close approach, the collision probability, the on-board available propellant mass and the damage level of the spacecraft. In the complex state also the dangerousness of the current orbit is considered. Concerning the action space, it is again divided into two different discrete models: the first is simpler and contains a set of five different impulses, corresponding to an altitude raised by multiples of 0.2 km; also a no manoeuvrer option is included. The second and more complex model considers a set of three

different impulses associated with an altitude increase or decrease by multiples of 0.3 km; a no manoeuvrer option is again included.

- **Reward Shaping.** Differently from the main methodology described in Section 6.1, the redundancy in the reward model has been sacrificed to benefit the variability of the state and action model. A series of objectives are accounted for in the reward definition: the distinction between dangerous and risk-free events, the effective reduction of the collision probability below a defined threshold, and the minimization of the damage suffered by the spacecraft and of the propellant.
- **Initial Conditions Robustness.** The complexity is divided into two subsets of environments: one case considers equatorial circular orbits with a radius equal to 7129 km; the second, instead, sun-synchronous circular orbits with a radius of 7078 km and inclination of 98.55°. This division is exploited to differentiate the training and sensitivity analysis, to test the robustness of the proposed approach.

As introduced before and summed up in Fig. 6.2, this application has been studied considering two different models (differentiated in the definition of the state-action spaces) and initial orbital conditions.

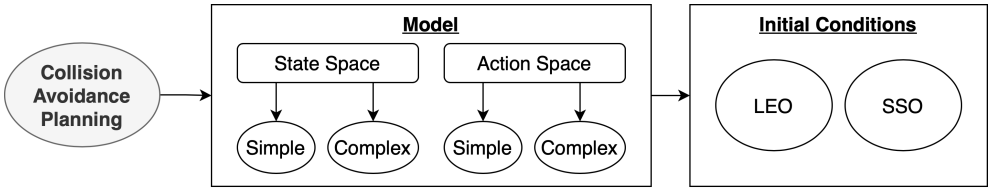


Figure 6.2: Collision avoidance scenario workflow scheme.

Orbit	State Space	Action Space	Avg. Result	
LEO at 7129 km	Simple	Simple	*Score* *Mass left* *Damage*	95 18 % 16 %
LEO at 7129 km	Complex	Complex	*Score* *Mass left* *Damage*	98 5 % 10 %
SSO at 7078 km	Complex	Complex	*Score* *Mass left* *Damage*	115 6 % 5 %

Table 6.1: Results of A2C-based collision avoidance strategy optimization for LEO satellites.

In Table 6.1 the results of the analysis performed in [109] are shown. As previously explained, they are divided by state and action spaces' complexity and orbit case. The scores outlined in the last column are referred to as baseline results of -50. In Fig. 6.3 and Fig. 6.4, two examples of the second and last cases are shown, depicting the trajectory and manoeuvrers' location, together with the evolution of spacecraft altitude, mass consumption, damage level and agent's score throughout the simulation. Each episode is simulated allocating 4kg of propellant for a time window of around 170h.

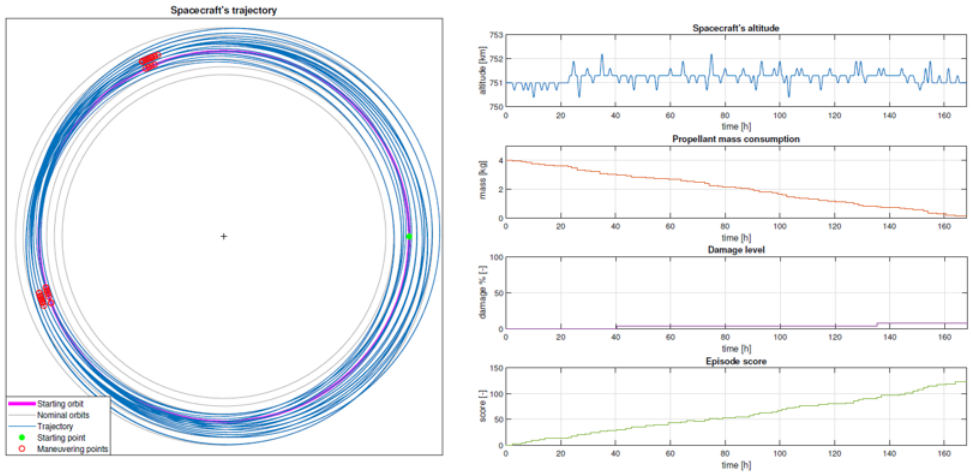


Figure 6.3: Example of simulation. Equatorial circular orbit of radius R=7029km, propellant mass available m=4kg [109].

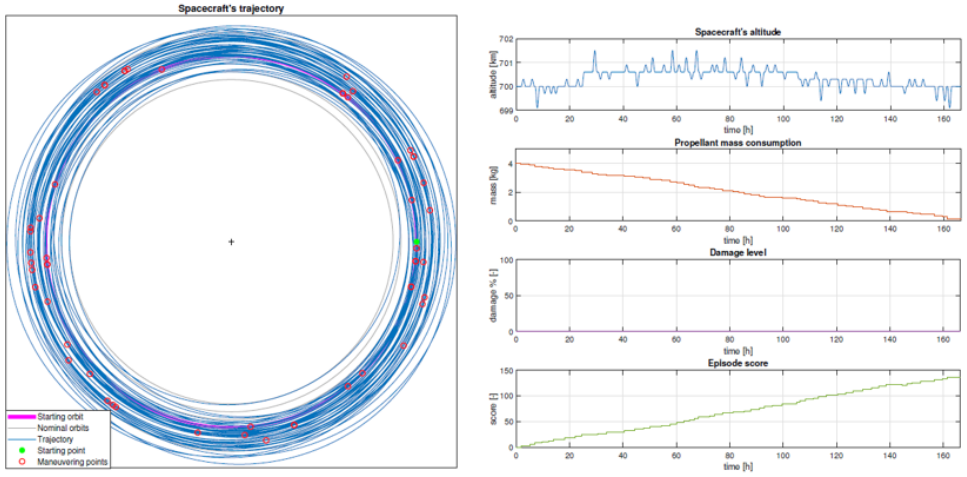


Figure 6.4: Example of simulation: this simulation mimics the on-board installation of a ready-to-use architecture trained on ground [109].

The reader, therefore, can easily notice how, similarly to what happened in the training phase of the main scenario treated in this thesis in Section 4.2, the proposed procedure of slowly increasing the complexity of the problem in terms of modelling and initial conditions robustness helps to shape the agent and get the desired performances.

6.2.2 Launcher's First Stage Propulsive Landing

***State-of-the-art* and Motivation.** In recent years, the planetary landing problem is gaining importance in the space sector, mostly pushed by the growing technology of reusable launchers: indeed, planetary landing spans a wide range of applications from unmanned probes landing on other planets or space bodies to reusable first and second stages of launcher vehicles. This problem wants to optimize the achievement of a successful touchdown on a planet in terms of time, and fuel consumption, minimizing the terminal error within prescribed location bounds, velocity limits, and attitude constraints. Since the first lunar landing rocket launched into space in the late 60s, several techniques have been developed, starting from simple fixed guidance solutions to advanced optimization techniques. The first application, experienced during the Apollo missions, exploited a polynomial trajectory guidance subdivided into different polynomials for each of the landing phases, reducing the need for the required computational resources [110]. During the decades, several improvements have been reached: from the definition of optimal guidance laws for fuel-efficient trajectories [111] to the latest Second Order Cone Programming methods [112] in which the non-convex environmental constraints have been relaxed by *convexification*. Moreover, even if this kind of methodology is limited by the facts that only linear dynamics can be optimized and not all the constraints can be convexified, this kind of solution has anyway extended to 6-DOF scenarios through the use of *successive convexification*, as studied in [113, 114]. Nowadays, the state-of-the-art approach requires the partition of the optimization process into sequential steps: starting from the computation of the reference trajectory exploiting the state estimation given by the navigation system, and then passing to the trajectory tracking through the use of a controller. To overcome this separation, different techniques aimed to unify and integrate guidance and control have been studied. As usual, one of the most promising and fascinating ones is related to the development of a DRL framework able to generate a landing policy directly mapping a full or partial observation of the state of the system to a control action, as applied to several guidance and control problems [6]. Some of the most recent works on DRL-based planetary landing investigate lunar rendezvous and landing using image-based guidance through meta-learning [36] or different thruster configurations for extraterrestrial planetary related problems [115, 116, 19]; they have generated promising results, although without any robustness analysis on unmodeled

dynamic or parametric uncertainties, and with limited dispersion in the initial mass. Therefore, it becomes crucial to assess the performance of these kinds of novel techniques and their advantages and disadvantages. This is one of the main pillars at the base of the analysis that will be presented here, which comes from a feasibility study performed in the framework of [117]. The work aims to apply model-free reinforcement learning to achieve successful atmospheric landing by directly controlling the launcher's gimbaled thruster, within the framework of an open-source, validated 6-DOF simulation environment. The training and testing strategy follows the methodology introduced in Section 6.1, investigating different levels of environment complexity, reward shaping and initial conditions.

Problem definition and Results. Recalling the scheme analysed before in Section 6.1, the most important aspects, related to the definition of this case study, can be summarized as follows:

- **Environment Complexity.** The problems have been subdivided into two levels of complexity: at first, the rocket model is studied on a 3-DOF basis defined in the inertial reference frame, under some peculiar assumption, i.e. atmospheric density constant, gravity field uniform, average inertia matrix, and axial aerodynamical force. In this case, the policy agent acts on the environment directly controlling the rocket thrust. The second level, instead, considers a full 6-DOF scenario in which the translational and rotational dynamics (governed by Euler's rigid body equations) are uncoupled. The previous assumptions still hold in this realistic case.

- **Model Architecture.** The adopted model falls again in the broad domain of active SLAM problems, characterized by a partially observable environment. Therefore, the usual methodology is applied making use of PPO to optimize the problem formulated with DRL. The architecture is based on a continuous state-action space. The state is defined by the rocket's position and velocity, angular orientation and velocity; the action, instead, consists of a thrust vector and gimbal angles with limited feasible ranges.

- **Reward Shaping.** A two-step reward function has been defined for both the two types of environment introduced before. While, at first, the reward targets an *heuristic* velocity, inspired by [19], aiming for the landing state, the second focuses more on the optimization of the fuel consumption [116].

- **Initial Conditions Robustness.** Two sets of initial conditions have been used for the training analysis; they are differentiated not by the randomness but by the complexity. The first set is *simplified* in order

to help with the tuning of the reward function; the second, instead, is derived from real rockets' reentry historical data.

In Fig. 6.5 the testing methods are schematized, partially reflecting the workflow proposed in the previous section. Indeed, in this particular case study, the diversification of the reward model is sacrificed to benefit the analysis on different models and initial conditions, without burdening too much the entire analysis. All the combinations of these two aspects are, therefore, studied. For an in-depth overview of this case study, please refer to [117].

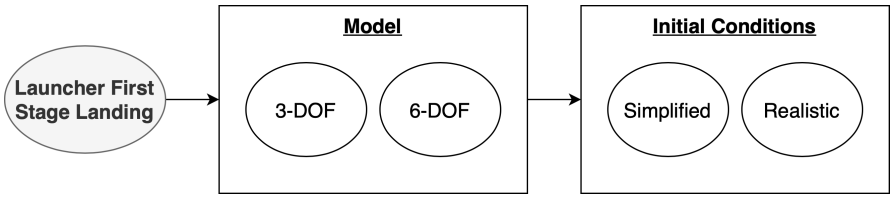


Figure 6.5: Launcher's first stage landing scenario workflow scheme.

Simplified Initial Conditions					
		position	*velocity*	*attitude*	-
3-DOF	μ	< 20 m	< 5 m/s	< 8 deg	-
Realistic Initial Conditions					
		position	*velocity*	*attitude*	-
3-DOF	μ	< 50 m	< 15 m/s	< 10 deg	-
Simplified Initial Conditions					
		position	*velocity*	*attitude*	*angular vel.*
6-DOF	μ	25.1 m	6.78 m/s	3.6 deg	0.03 deg/s
	σ	3.5 m	1.2 m/s	2.0 deg	0.0 deg/s
Realistic Initial Conditions					
		position	*velocity*	*attitude*	*angular vel.*
6-DOF	μ	10.0 m	9.42 m/s	4.7°	0.06 deg/s
	σ	4.3 m	2.3 m/s	2.7 deg	0.04 deg/s

Table 6.2: Results of PPO-based guidance optimization for rocket's first stages reentry landing.

The training and testing results are summarized in Table 6.2, showing them in terms of the 3 and 6-DOF models, combined with the simplified and realistic initial conditions. The procedure mirrors the same development used in the main methodology. Fig. 6.6 shows the evolution of the terminal position and velocity errors throughout the training simulation for the 6-DOF under realistic

initial conditions, together with an example of the resulting trajectory, which displays the braking rate of the velocity during the landing.

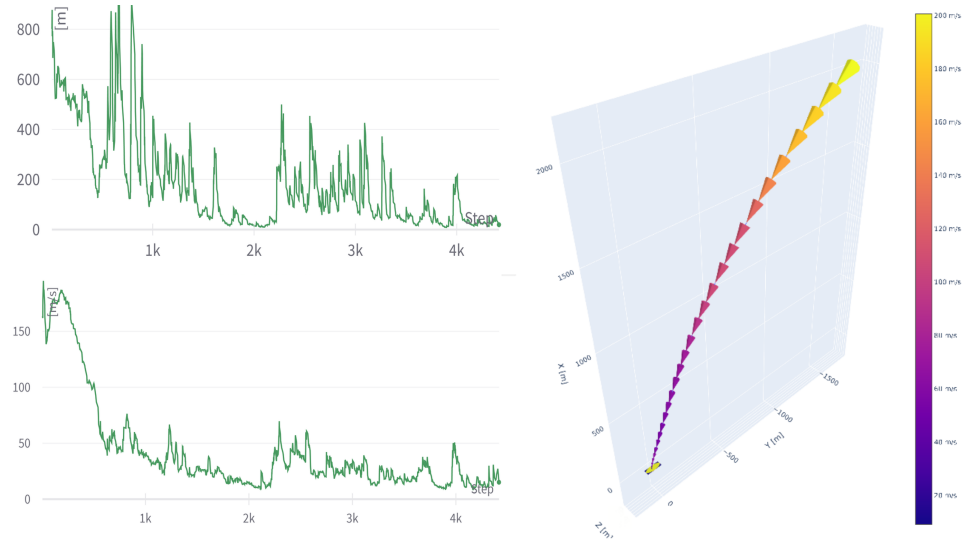


Figure 6.6: Example of terminal position and velocity errors through the training simulation, and out-coming trajectory [117].

Once again, the results show that it is possible to achieve good performances also for very complex models and environmental conditions thanks to an incremental and programmatic way of raising the complexity of the scenario.

6.2.3 Robotic Capture of Uncooperative Targets

***State-of-the-art* and Motivation.** Following the research inputs already discussed for the main scenario (spacecraft relative guidance) and the first of the three scenarios presented in this chapter, this work falls again in the context of In-Orbit Servicing (IOS) and Active Debris Removal (ADR), which, as already underlined, are operations of increasing interest, especially to guarantee the longevity of space endeavours and reduce the costs of the missions. In this framework, several strategies have been proposed to reduce the active debris problem, mainly focusing on nets and tethered systems, harpoons to capture the debris or drag augmentation devices, and ion beams to push them out of orbit [118]. These kinds of approaches are suited for ADR but are not applicable to IOS; reason why, instead, the interest in robotic systems is strongly increased. The problem with these systems is that their complexity is such high that, up to now, any sort of automation has been deprecated. Indeed, the first space manipulators, such as Canadarm2 and the European Robotic Arm (ERA), were designed to be remotely controlled by humans [119]. In the last decades, several studies have been performed on GNC methods for floating-

base robotic arm control, from virtual model and Generalized Jacobian Matrix (GJM) control approaches [120, 121] to optimal control, heuristic methods, i.e genetic algorithm, and adaptive control techniques [122, 123]. In the end, the application of DRL to enhance the guidance and control of robotic arms in space, emerges as a potential solution thanks to its main features, which may help to overcome problems such as the inverse kinematics for high-DOF manipulators, collision avoidance and motion constraints. The most interesting studies have been performed on trajectory planning of 6-DOF manipulator for simple end-effector positioning and attitude alignment objective in [124], path-planning of a 3-DOF robotic arm with obstacle avoidance in [125], or 6-DOF manipulator position tracking of circular trajectories with the end-effector, as done in [126]. The application briefly described here is extrapolated by the wider framework developed in [127], where, through the use of DRL, is wanted to train and test an autonomous agent to acquire the capability of carrying out the guidance tasks of a robotic manipulator, to synchronize the end-effector's state (position + attitude) to grasp the uncooperative target, and finally reducing any relative motion.

Problem definition and Results. Also in this application, the workflow defined in Fig. 6.1 has been partially followed, in particular in terms of reward shaping and initial conditions analysis. The most important aspects of the problem definition are here listed:

- **Environment Complexity.** The problem is based on a space robot, which is modelled as a multi-body system with a 6-DOF base and a 7-DOF manipulator. The robot is assumed to be free-flying, with a base that can be actively controlled when needed. The robot kinematics is defined by a direct-path approach, exploiting Newton-Euler dynamics formulation [128]. On the other hand, the target is centred on the LVLH reference frame's origin, thus neglecting its translational dynamics, and formulating its attitude with Euler's equations of motion; it is shaped as a cylinder with two solar panels as shown in Fig. 6.8.

- **Model Architecture.** The problem is optimized by making use of the PPO algorithm. The input state is composed of the current joint angles and rates, the errors between the end-effector's desired and current states, and the relative distances between the manipulator's links and the target. The action space, instead, is discrete and it is defined by the desired joint rates of the manipulator, which are fed to the robotic arm controller, which is a simple tuned PD. Throughout the entire analysis summarized here, this model is always kept constant.

- **Reward Shaping.** The reward function is based on three main objectives: the position and attitude of the end-effector guidance to the

desired state, the minimization of the time taken by the end-effector to synchronize with the desired state, and the avoidance of a keep-out zone around the target. Two different models have been defined: the first only takes into consideration the first two objectives, and the second, instead, adds also the collision avoidance one.

- **Initial Conditions Robustness.** Different levels of initial conditions are defined in the analysis framework: the basic one features the fixed position of the centre of mass of the robot's base along the target's angular momentum vector as defined in the motion synchronization approach developed in [129]; moreover, the target's grasping point is randomized but limited to the face shape's perimeter, and the manipulator's initial configuration is randomized at each episode. The second case of initial conditions, instead, is more complex: the initial manipulator configuration maintains the same level of randomness, but a further level is reached considering that the target spin velocity is randomized between a limited range, a random displacement is added to the robot's base-target distance, and also the grasping position is extended to the whole face of the target.

Fig. 6.7 sums up the aspects which may vary during the training and testing analysis of this case study.

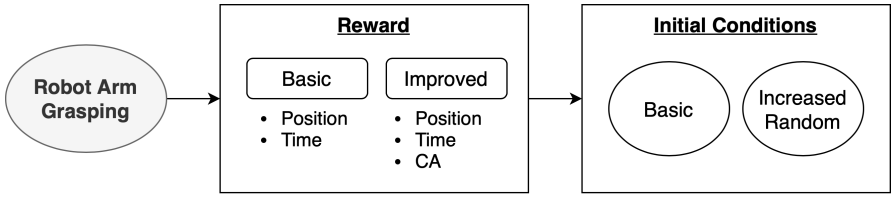


Figure 6.7: Robotic arm grasping scenario workflow scheme.

In Fig. 6.8, the scenario involving the target object and the chaser equipped with the robotic arm is represented.

In Table 6.3, the main results obtained during the analysis have been summarized. It can be noticed how the workflow philosophy has been followed differentiating between different reward models and initial conditions values. The first analysis, indeed, is performed with an agent trained with a reward defined for the position and time objectives and then tested for guidance and time. Afterwards, the complexity is increased, adding the collision avoidance (CA) objective to the reward model. Once trained, the agent is also tested for different levels of random initial conditions. Since the performances were too low both for random chaser-target relative distance and grasping point, a transfer learning approach has been adopted to recover the optimal level of performance, training the new agent with random initial conditions.

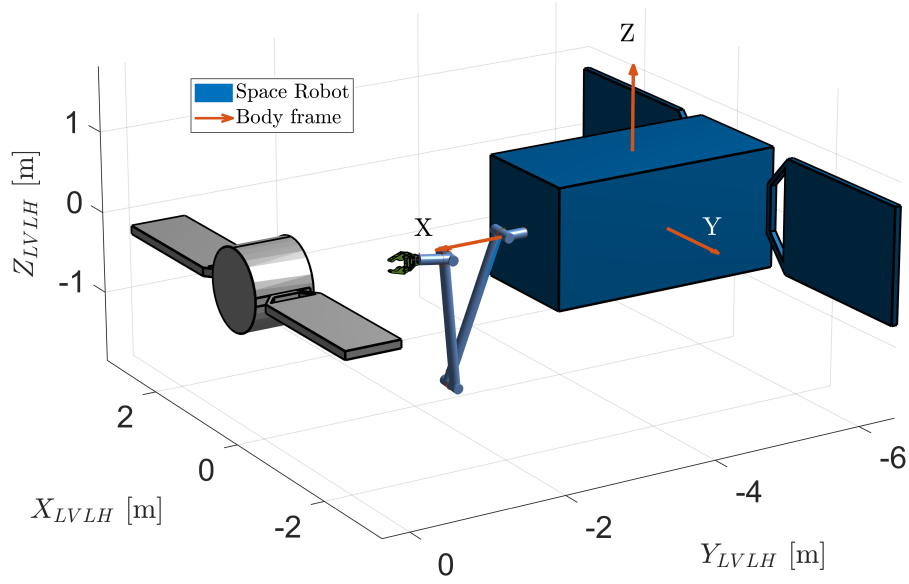


Figure 6.8: Representation of the environment for the in-orbit servicing motion synchronization and grasping scenario [127].

Sim.	Reward	Objective	Init.l conditions	Success Rate
Train	Pos.+Time	GC	Basic	93.5 %
Test	Pos.+Time	Time	Basic	100 %
Train	Pos.+Time+CA	GC+CA	Basic	100 %
Test	Pos.+Time+CA	GC+CA	Dis. Random	75 %
Test	Pos.+Time+CA	GC+CA	Grasp. Random	47 %
Train	Pos.+Time+CA	GC+CA	Random	100 %

Table 6.3: Robotic arm autonomous guidance results.

Once more the process of gradually raising the width of the problem, adding complexity step by step to the reward model and to the initial conditions, brings benefit to the development of the autonomous agent capable of satisfying correctly the wanted objectives.

6.2.4 Closing remarks

All the presented case studies support the main analysis carried out throughout the previous chapters of this thesis. All of them helped to tune the methodology that, therefore, is proposed to investigate the performance and the capability of the DRL-based approach. In conclusion, it is interesting to notice the location and moment when these new scenarios have been analysed along the

development of the main research's path. In particular, as shown in Fig. 1.1, the first case treating CAM strategy planning, is located at the same starting point of the main relative dynamics scenario. Indeed, this side work helped to strengthen the feasibility of reinforcement learning as a valuable tool for planning problems. The second case, the one related to the launcher's first stage landing guidance, is instead collocated at the end of the main scenario baseline design and analysis; therefore, once the main methodology was already robust enough to be confidently applied to a different kind of environment, which is anyway characterized by the same type of models and variability. That is why, the resulting analysis, performed on a continuous state-action space, backs up the choice of going more in-depth in the design of more complex state and action spaces, defining them more closely to a spacecraft's realistic behaviour. Finally, the last case concerns the robotic arm guidance for uncooperative target grasping and is positioned in a timeline where the methodology had already a great level of robustness to be able to perform all sorts of sensitivity analysis. For this reason, indeed, the analysis of this case study pushes especially in the direction of understanding all the possible failures derived by a training activity which is inevitably unaware of all the aspects which may affect the knowledge of the environment. In the end, these side-analyses on different space environments and applications are judged helpful and of completion to the main research objectives achieved in the thesis.

CHAPTER 7

Conclusion

> E qui l'eroiche gesta trovano una chiosa,
> più che in poemetto, dovrebbero essere in
> prosa.
> Non temete, oh lettori, breve sarà l'attesa,
> che il Brando già si prepara, per una
> nuova impresa.
>
> — Manuel Indaco, 02/10/2022

In conclusion, this chapter wants to summarize the overall results obtained throughout all the thesis work and to wrap up an overview of all the advancement that this thesis tried to bring in the context of autonomous guidance planning for spacecraft in relative motion around uncooperative and unknown target objects. This problem belongs to the broad domain of autonomous On-Orbit Servicing applications aimed to increase the independence of the spacecraft from human intervention in order to enhance the reliability, cost-effectiveness, and flexibility of missions' planning, lowering the overall risks. This incredible boost that pushes space-related research towards autonomous systems, is nowadays possible especially because of the great developments

on AI and machine learning, which is the second main backbone of the work carried out through this Ph.D..

In summary, the work deals with an adaptive guidance algorithm aimed to autonomously reconstruct the geometrical shape of an uncooperative artificial object in space. The fly-around and mapping problem has been mathematically formulated as a Partially Observable Markov Decision Process (POMDP) in the framework of autonomous exploration active SLAM problem. Relative dynamics and visibility models dictate the outcome of the simulations, describing how the chaser moves around the target and which surfaces are visible from the camera (whether it is VIS or TIR), at the beginning under the assumption that they are always pointing towards the target object. When vision-based architecture is assumed, the simulations employ images in the visible band and process them to obtain the object map via SPC guidelines. This step generates the essential inputs for the following guidance block, which has been implemented with the state-of-the-art DRL algorithm for continuous state-action spaces, namely Proximal Policy Optimization (PPO), to design the decision-making policy of the spacecraft agent. The training of the PPO agent is then studied and analysed, investigating two different network models, characterized by linear or recurrent architectures respectively, followed by an extensive training campaign to assess the model robustness and sensitivity in terms of reward models and random initial conditions. Afterwards, two of the main model's assumptions - *discrete* action space and *perfect* input state - have been relaxed, studying the resulting implications. On the one hand, a continuous action space is analysed, proving that this DRL-based methodology is completely feasible to obtain optimal performances even with a massively more complex action space design. The second analysis instead aimed to analyse the case of uncertainty in the input state vector and how the associated noise affects the performance of the algorithm. At first, results showed that the main models were not robust to the input state affected by noise. Therefore, two different methods have been proposed and investigated: a re-training procedure and a transfer-learning procedure. Both of them improve the capabilities of the autonomous guidance agent, even if, it is also demonstrated how transfer-learning shows up more robust to further change in the input noise. Therefore, this part of the work confirmed and developed further the applicability of DRL algorithms to the spacecraft autonomous guidance problem, applied for the shape reconstruction of an uncooperative target.

Once the baseline model has been deeply analysed, an entire image and AI-based GNC algorithm architecture has been proposed and presented in all of its components. The guidance and control block is tested through an extensive sensitivity analysis, shaping the environment on the TANGO object. Afterwards, the agent's level of robustness and flexibility was assessed, and the developed block was inserted and interfaced with the image-based

navigation tool. Although some of those were quite distant with respect to the environmental model in which the agent itself has been trained, the agent exhibits stability and unaltered behaviour in terms of level of performance throughout all the defined steps of the testing simulations. The policy learned seems to be robust to all the uncertainties and model differences introduced in the new GNC framework, how it is demonstrated by the trajectory path that the agent follows, or at least tries to follow, at each simulated episode. In conclusion, even if some points still remain open, e.g. coupling compatibility between trajectory and attitude control, absolute attitude knowledge assumption, etc., the main objective of this analysis, is achieved with good success. Of course, these results have strengthened the awareness of having a powerful tool, that, at this point, feels the need of being tested also in a realistic or hardware-in-the-loop environment.

At the end of the journey, it has been decided to underline not only the potentiality of the innovative mathematical approach on a particular space-related scenario but also the strengths of the methodology followed during the training and testing processes. To investigate this point, the same method has been followed, tailoring the pipeline for each of the three presented case studies: strategy planning for debris collision avoidance manoeuvrers, launcher's first stage autonomous propulsive lading, and robotic capture of uncooperative targets. These different applications have been analysed along the development of the main research topic, corroborating its procedures and techniques. All three cases confirmed the good quality of the proposed methodology, achieving satisfying performance levels in each of the applications' frameworks.

Working for a Ph.D. is not an easy road, it is a path full of stops, sudden turns, backs out and comebacks, and surely does not exhaustively explore a research field. But certainly, it helps pushing towards the accomplishment of evermore exciting goals. That is why, before concluding this Ph.D. dissertation, it may be interesting to briefly mention some of the possible future developments and open points which inevitably arise after such kind of highway.

7.1 Future Steps

Some interesting future developments are worth introducing shortly. As outlined throughout the overall thesis, every deep reinforcement learning agent needs to pursue the stability of its model and the capability of facing environments which may not be exactly equal to the one it is trained on. Among all, this concept is valid for state uncertainty, action limits or space, mission objectives and so on. For this reason, two main directions are suggested for the continuation of this research topic: one is related to the model, and the second, instead, concerns the testing framework.

Model improvement. As briefly mentioned at the end of Chapter 4, it could be strongly beneficial for the improvement of the agent robustness to integrate the proposed network architecture with a model-based network. This may help also in treating and implementing the chaser attitude control, which up to now is something that the agent is never been trained on. A model of the chaser attitude dynamics would bring much closer to reality the discussed method, making sure that the map level during a single episode increases only when the chaser is correctly pointing toward the target and can take pictures of only the faces that are inside the field of view of the cameras. From the RL side, this development could be managed in different ways, for example:

- implementing the attitude dynamics and controlling it concurrently with the orbital dynamics, so that the same agent outputs the action vector containing both attitude and orbital controls;
- implementing the attitude dynamics and developing a multi-agent reinforcement learning methodology. In this case, one agent would be responsible for the orbital motion and the other for the attitude, meaning that both of them have to learn how to cooperate to reach the end goal of mapping the target object.

Testing framework. Moreover, as treated in particular in Chapter 5, this research is directed towards the coupling of the proposed guidance and control block with an AI-based navigation tool, which exploits simulated o real images as input to retrieve the relative pose. Therefore, one of the more interesting options to further investigate the entire GNC pipeline is certainly a hardware-in-the-loop analysis. This is a complex path, which may also require specific facilities but, at the same time, may be greatly useful in understanding the real applicability of such algorithms. The activity may be subdivided into two parallel analyses: one aimed to study and optimize the breadboarding of the image processing, navigation, guidance and control algorithms, the other more related to the development of a testing facility equipped with a target object mock-up, an illumination system and a VIS/TIR camera device which may simulate the chaser behaviour if placed on a controllable robotic arm. Thanks to this twofold research, it would be possible to concretely evaluate the performance of a realistic image and AI-based GNC system.

These are just two, among dozens of paths which may be followed to hold alive this field of study. This is the beauty and the challenge of the *Research*: small resources and endless problems. The chance of success is very narrow, but still, we keep looking forward to it.

Bibliography

[1] A. Flores-Abad, O. Ma, K. Pham, and S. Ulrich, "A review of space robotics technologies for on-orbit servicing", *Progress in Aerospace Sciences*, vol. 68, pp. 1–26, 2014, ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2014.03.002.

[2] M Shan, J Guo, and E. Gill, "Review and comparison of active space debris capturing and removal methods", English, *Progress in Aerospace Sciences*, vol. 80, no. January, pp. 18–32, 2016, ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2015.11.001.

[3] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control", *Astrodynamics*, vol. 3, pp. 287–299, 2019.

[4] M. Tipaldi, R. Iervolino, and P. R. Massenio, "Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges", *Annual Reviews in Control*, vol. 54, pp. 1–23, 2022, ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2022.07.004.

[5] K. Hovell and S. Ulrich, "Deep reinforcement learning for spacecraft proximity operations guidance", *Journal of Spacecraft and Rockets*, vol. 58, no. 2, pp. 254–264, 2021. DOI: https://doi.org/10.2514/1.A34838.

[6] S. Silvestrini *et al.*, "Chapter fifteen - modern spacecraft gnc", in *Modern Spacecraft Guidance, Navigation, and Control*, V. Pesce, A. Colagrossi, and S. Silvestrini, Eds., Elsevier, 2023, pp. 819–981, ISBN: 978-0-323-90916-7. DOI: https://doi.org/10.1016/B978-0-323-90916-7.00015-9.

[7] D. Izzo, G. Meoni, P. Gómez, D. Dold, and A. Zoechbauer, *Selected trends in artificial intelligence for space applications*, 2022. arXiv: 2212.06662 [cs.LG].

[8] S. Silvestrini *et al.*, "Optical navigation for lunar landing based on convolutional neural network crater detector", *Aerospace Science and Technology*, vol. 123, p. 107 503, 2022, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2022.107503.

[9] S. Silvestrini, M. Piccinin, G. Zanotti, A. Brandonisio, P. Lunghi, and M. Lavagna, "Implicit extended kalman filter for optical terrain relative navigation using delayed measurements", *Aerospace*, vol. 9, no. 9, 2022, ISSN: 2226-4310. DOI: 10.3390/aerospace9090503.

[10] V. Pesce, S. Silvestrini, and M. Lavagna, "Radial basis function neural network aided adaptive extended kalman filter for spacecraft relative navigation", *Aerospace Science and Technology*, vol. 96, p. 105 527, 2020, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2019.105527.

[11] S. Silvestrini and M. Lavagna, "Neural-based predictive control for safe autonomous spacecraft relative maneuvers", *Journal of Guidance Control and Dynamics*, vol. 44, pp. 2303–2310, Sep. 2021. DOI: https://doi.org/10.2514/1.G005481.

[12] S. Silvestrini and M. Lavagna, "Neural-aided gnc reconfiguration algorithm for distributed space system: Development and pil test", *Advances in Space Research*, vol. 67, no. 5, pp. 1490–1505, 2021, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2020.12.014.

[13] V. Pesce, A.-a. Agha-mohammadi, and M. Lavagna, "Autonomous navigation and mapping of small bodies", in *2018 IEEE Aerospace Conference, Big Sky, Montana, USA*, 2018, pp. 1–10. DOI: https://doi.org/10.1109/AERO.2018.8396797.

[14] D. M. Chan and A.-a. Agha-mohammadi, "Autonomous imaging and mapping of small bodies using deep reinforcement learning", in *2019 IEEE Aerospace Conference, Big Sky, Montana, USA*, 2019, pp. 1–12. DOI: https://doi.org/10.1109/AERO.2019.8742147.

[15] M. Piccinin, P. Lunghi, and M. Lavagna, "Deep reinforcement learning-based policy for autonomous imaging planning of small celestial bodies mapping", *Aerospace Science and Technology*, vol. 120, p. 107 224, 2022, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2021.107224.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[17] B. Gaudet, R. Linares, and R. Furfaro, "Adaptive guidance and integrated navigation with reinforcement meta-learning", *Acta Astronautica*, vol. 169, pp. 180–190, 2020, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2020.01.007.

[18] B. Gaudet, R. Linares, and R. Furfaro, "Six degree-of-freedom body-fixed hovering over unmapped asteroids via lidar altimetry and reinforcement meta-learning", *Acta Astronautica*, vol. 172, pp. 90–99, 2020, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2020.03.026.

[19] B. Gaudet, R. Linares, and R. Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing", *Advances in Space Research*, vol. 65, no. 7, pp. 1723–1741, 2020, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2019.12.030.

[20] A. Scorsoglio, R. Furfaro, R. Linares, and M. Massari, "Relative motion guidance for near-rectilinear lunar orbits with path constraints via actor-critic reinforcement learning", *Advances in Space Research*, vol. 71, no. 1, pp. 316–335, 2023, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2022.08.002.

[21] L. Federici, B. Benedikter, and A. Zavoli, "Deep learning techniques for autonomous spacecraft guidance during proximity operations", *Journal of Spacecraft and Rockets*, vol. 58, no. 6, pp. 1774–1785, 2021. DOI: https://doi.org/10.2514/1.A35076.

[22] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i", *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. DOI: https://doi.org/10.1109/MRA.2006.1638022.

[23] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping (slam): Part ii", *IEEE Robotics Automation Magazine*, no. 3, pp. 108–117, 2006. DOI: 10.1109/MRA.2006.1678144.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. Cambridge, Massachusetts, USA: MIT press, 2018.

[25] S. Silvestrini and M. Lavagna, "Deep learning and artificial neural networks for spacecraft dynamics, navigation and control", *Drones*, vol. 6, no. 10, p. 270, 2022.

[26] A. Brandonisio, *Deep reinforcement learning to enhance fly-around guidance for uncooperative space objects smart imaging*, Jul. 2020.

[27] A. Colagrossi, S. Silvestrini, and V. Pesce, "Chapter two - reference systems and planetary models", in *Modern Spacecraft Guidance, Navigation, and Control*, V. Pesce, A. Colagrossi, and S. Silvestrini, Eds., Elsevier, 2023, pp. 45–75, ISBN: 978-0-323-90916-7. DOI: https://doi.org/10.1016/B978-0-323-90916-7.00002-0.

[28] A. Capannolo, S. Silvestrini, A. Colagrossi, and V. Pesce, "Chapter four - orbital dynamics", in *Modern Spacecraft Guidance, Navigation, and Control*, V. Pesce, A. Colagrossi, and S. Silvestrini, Eds., Elsevier, 2023, pp. 131–206, ISBN: 978-0-323-90916-7. DOI: https://doi.org/10.1016/B978-0-323-90916-7.00004-4.

[29] H. D. Curtis, *Orbital Mechanics for Engineering Students*. Butterworth-Heinemann, 2014.

[30] W. H. Clohessy and R. S. Wiltshire, "Terminal guidance system for satellite rendezvous", *Journal of the Aerospace Sciences*, vol. 27, no. 9, pp. 653–658, 1960. DOI: https://doi.org/10.2514/8.8704.

[31] J. R. Wertz, *Spacecraft Attitude Determination and Control*. Springer Science & Business Media, 1978.

[32] G. Ciabatti, S. Daftry, and R. Capobianco, "Autonomous planetary landing via deep reinforcement learning and transfer learning", in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 2031–2038. DOI: 10.1109/CVPRW53098.2021.00231.

[33] A. Brandonisio, M. Lavagna, and D. Guzzetti, "Reinforcement learning for uncooperative space objects smart imaging path-planning", *The Journal of the Astronautical Sciences*, vol. 68, no. 4, 1145–1169, Nov. 2021. DOI: https://doi.org/10.1007/s40295-021-00288-7.

[34] "Deep reinforcement learning spacecraft guidance with state uncertainty for autonomous shape reconstruction of uncooperative target", *Advances in Space Research*, 2023, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2023.07.007.

[35] G. Barth-Maron *et al.*, "Distributed distributional deterministic policy gradients", *CoRR*, vol. abs/1804.08617, 2018. arXiv: 1804.08617.

[36] A. Scorsoglio, A. D'Ambrosio, L. Ghilardi, B. Gaudet, F. Curti, and R. Furfaro, "Image-based deep reinforcement meta-learning for autonomous lunar landing", *Journal of Spacecraft and Rockets*, vol. 59, no. 1, pp. 153–165, 2022. DOI: 10.2514/1.A35072. eprint: https://doi.org/10.2514/1.A35072.

[37] B. Gaudet, R. Linares, and R. Furfaro, "Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations", *Acta Astronautica*, vol. 171, pp. 1–13, 2020, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2020.02.036.

[38] H. Li, Q. Gao, Y. Dong, and Y. Deng, "Spacecraft relative trajectory planning based on meta-learning", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 57, no. 5, pp. 3118–3131, 2021. DOI: 10.1109/TAES.2021.3071226.

[39] R. Bellman, "A markovian decision process", *Journal of Mathematics and Mechanics*, 1957.

[40] R. A. Howard, *Dynamic Programming and Markov Processes*. The M.I.T. Press, 1960.

[41] K. Åström, "Optimal control of markov processes with incomplete state information", *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 174 –205, 1965. DOI: https://doi.org/10.1016/0022-247X(65)90154-X.

[42] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains", *Artificial Intelligence*, 1998. DOI: https://doi.org/10.1016/S0004-3702(98)00023-X.

[43] G. Shani, J. Pineau, and R Kaplow, "A survey of point-based pomdp solvers", *Auton Agent Multi-Agent Syst*, 2012. DOI: https://doi.org/10.1007/s10458-012-9200-2.

[44] M. A. Nielsen, *Neural Network and Deep Learning*. Determination Press, 2015.

[45] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, Massachusetts, USA: MIT Press, 2016.

[46] "A focused backpropagation algorithm for temporal pattern recognition", *Complex Systems*,

[47] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[48] H. Sak, A. Senior, and F. Beaufays, *Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition*, 2014. arXiv: 1402.1128 [cs.NE].

[49] J. Brownlee, *Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions*. Machine Learning Mastery, 2019.

[50] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[51] J. Read, A. Bifet, B. Pfahringer, and G. Holmes, "Batch-incremental versus instance-incremental learning in dynamic and evolving data", in *Advances in Intelligent Data Analysis XI*, J. Hollmén, F. Klawonn, and A. Tucker, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 313–323.

[52] A. T. Azar *et al.*, "Drone deep reinforcement learning: A review", *Electronics*, vol. 10, no. 9, 2021, ISSN: 2079-9292. DOI: 10.3390/electronics10090999.

[53] V. Mnih *et al.*, *Asynchronous methods for deep reinforcement learning*, 2016. arXiv: 1602.01783 [cs.LG].

[54] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization", in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, 2015, pp. 1889–1897.

[55] H. Shalma and P. Selvaraj, "A review on 3d image reconstruction on specific and generic objects", *Materials Today: Proceedings*, vol. 80, pp. 2400–2405, 2023, SI:5 NANO 2021, ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2021.06.371.

[56] A. Kundu, Y. Li, and J. M. Rehg, "3d-rcnn: Instance-level 3d object reconstruction via render-and-compare", in *2018 IEEE/CVF Conference*

on Computer Vision and Pattern Recognition, 2018, pp. 3559–3568. DOI: 10.1109/CVPR.2018.00375.

[57] X.-F. Han, H. Laga, and M. Bennamoun, "Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1578–1604, 2021. DOI: 10.1109/tpami.2019.2954885.

[58] H. Zhang, Q. Wei, and Z. Jiang, "3d reconstruction of space objects from multi-views by a visible sensor", *Sensors*, vol. 17, no. 7, 2017, ISSN: 1424-8220. DOI: 10.3390/s17071689.

[59] B. Caruso, T. Mahendrakar, V. M. Nguyen, R. T. White, and T. Steffen, *3d reconstruction of non-cooperative resident space objects using instant ngp-accelerated nerf and d-nerf*, 2023. arXiv: 2301.09060 [cs.CV].

[60] R. W. Gaskell, "Automated landmark identification for spacecraft navigation", *Advances in the Astronautical Sciences*, vol. 109, pp. 1749–1756, 2001.

[61] R. W. Gaskell *et al.*, "Characterizing and navigating small bodies with imaging data", *Meteoritics & Planetary Science*, vol. 43, no. 6, pp. 1049–1061, 2008. DOI: 10.1111/j.1945-5100.2008.tb00692.x.

[62] M. Bechini, M. Lavagna, and P. Lunghi, "Dataset generation and validation for spacecraft pose estimation via monocular images processing", *Acta Astronautica*, vol. 204, pp. 358–369, 2023, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2023.01.012.

[63] G. L. Civardi, M. Bechini, M. Quirino, A. Colombo, M. Piccinin, and M. Lavagna, "Generation of fused visible and thermal-infrared images for uncooperative spacecraft proximity navigation", *Advances in Space Research*, 2023, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2023.03.022.

[64] S. Silvestrini, J. Prinetto, G. Zanotti, and M. Lavagna, "Design of robust passively safe relative trajectories for uncooperative debris imaging in preparation to removal", in *2020 AAS/AIAA Astrodynamics Specialist Conference, Lake Tahoe, California, USA*, Lake Tahoe, California, USA, Aug. 2020.

[65] E. Gill, S. D'Amico, and O. Montenbruck, "Autonomous formation flying for the prisma mission", *Journal of Spacecraft and Rockets*, vol. 44, no. 3, pp. 671–681, 2007. DOI: 10.2514/1.23015. eprint: https://doi.org/10.2514/1.23015.

[66] L. Capra, "Deep reinforcement learning towards adaptive vision-based autonomous guidance", Master's Thesis, Politecnico di Milano, 2020-2021.

[67] J. Martínez, D. Rafalskyi, and A. Aanesland, "Development and testing of the npt30-i2 iodine ion thruster", in *36th International Electric Propulsion Conference*, Vienna, Austria, Sep. 2019. DOI: 10.6084/m9.figshare.11931363.

[68] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks", *International Conference on Learning Representations*, 2014. arXiv: 1312.6120 [cs.NE].

[69] L. Capra, A. Brandonisio, and M. Lavagna, "Network architecture and action space analysis for deep reinforcement learning towards spacecraft autonomous guidance", *Advances in Space Research*, vol. 71, no. 9, pp. 3787–3802, 2023, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2022.11.048.

[70] R. Biesbroek, S. Aziz, A. Wolahan, S. Cipolla, M. Richard-Noca, and L. Piguet, "The clearspace-1 mission: Esa and clearspace team up to remove debris", in *Proceedings of 8th European Conference on Space Debris*, 2021, pp. 1–3.

[71] J. Sullivan, S. Grimberg, and S. D'Amico, "Comprehensive survey and assessment of spacecraft relative motion dynamics models", *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 8, pp. 1837–1859, 2017. DOI: https://doi.org/10.2514/1.G002309.

[72] G. Xu and D. Wang, "Nonlinear dynamic equations of satellite relative motion around an oblate earth", *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 5, pp. 1521–1524, 2008. DOI: https://doi.org/10.2514/1.33616.

[73] M. Piazza, M. Maestrini, and P. Di Lizia, "Monocular relative pose estimation pipeline for uncooperative resident space objects", *Journal of Aerospace Information Systems*, vol. 19, no. 9, pp. 613–632, 2022. DOI: https://doi.org/10.2514/1.I011064.

[74] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7559–7566. DOI: 10.1109/ICRA.2018.8463189.

[75] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning", *Journal of Big Data*, vol. 3, no. 1, pp. 1–40, 2016. DOI: 10.1186/s40537-016-0043-6.

[76] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman, "Efficient adaptation for end-to-end vision-based robotic manipulation", *CoRR*, vol. abs/2004.10190, 2020. arXiv: 2004.10190.

[77] F. Zhuang *et al.*, "A comprehensive survey on transfer learning", *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021. DOI: 10.1109/JPROC.2020.3004555.

[78] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 13 344–13 362, 2023. DOI: 10.1109/TPAMI.2023.3292075.

[79] M. Bechini, "Monocular vision for uncooperative targets through ai-based methods and sensors fusion", Ph.D. Dissertation, Politecnico di Milano, 2024.

[80] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. MIT Press, 2023.

[81] J. Guarneri, *Autonomous optical navigation and attitude estimation system tested on artificially generated images*, Jul. 2020.

[82] P. Lunghi, M. Ciarambino, and M. Lavagna, "A multilayer perceptron hazard detector for vision-based autonomous planetary landing", *Advances in Space Research*, vol. 58, no. 1, pp. 131–144, 2016, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2016.04.012.

[83] A. Rivolta, P. Lunghi, and M. Lavagna, "Gnc & robotics for on orbit servicing with simulated vision in the loop", *Acta Astronautica*, vol. 162, pp. 327–335, 2019, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2019.06.005.

[84] N. Otsu, "A threshold selection method from gray-level histograms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. DOI: 10.1109/TSMC.1979.4310076.

[85] M. Bechini, P. Lunghi, and M. Lavagna, "Tango Spacecraft Dataset for Monocular Pose Estimation", Zenodo, Apr. 2022. DOI: 10.5281/zenodo.6499007.

[86] S. Sharma, J. Ventura, and S. D'Amico, "Robust model-based monocular pose initialization for noncooperative spacecraft rendezvous", *Journal of Spacecraft and Rockets*, vol. 55, no. 6, pp. 1414–1429, 2018. DOI: 10.2514/1.A34124.

[87] M. Kisantal, S. Sharma, T. H. Park, D. Izzo, M. Märtens, and S. D'Amico, "Satellite pose estimation challenge: Dataset, competition design, and results", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 5, pp. 4083–4098, 2020.

[88] E. M. V. Association, "Emva 1288 standard: Standard for characterization of image sensors and cameras", in *EMVA, Tech. Rep. Release 4.0 Linear*, 2021.

[89] M. Konnik and J. Welsh, "High-level numerical simulations of noise in ccd and cmos photosensors: Review and tutorial", *arXiv preprint arXiv:1412.4031*, 2014.

[90] K. Wei, Y. Fu, Y. Zheng, and J. Yang, "Physics-based noise modeling for extreme low-light photography", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 8520–8537, 2022. DOI: 10.1109/TPAMI.2021.3103114.

[91] T. H. Park *et al.*, "Satellite pose estimation competition 2021: Results and analyses", *Acta Astronautica*, vol. 204, pp. 640–665, 2023, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2023.01.002.

[92] T. H. Park and S. D'Amico, "Robust multi-task learning and online refinement for spacecraft pose estimation across domain gap", *Advances in Space Research*, 2023, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2023.03.036.

[93] M. Kisantal, S. Sharma, T. H. Park, D. Izzo, M. Märtens, and S. D'Amico, "Spacecraft Pose Estimation Dataset (SPEED)", Zenodo, Feb. 2019. DOI: 10.5281/zenodo.6327546.

[94] M. Bechini, P. Lunghi, and M. Lavagna, "Tango Spacecraft Wireframe Dataset Model for Line Segments Detection", Zenodo, Mar. 2022. DOI: 10.5281/zenodo.6372849.

[95] M. Bechini, P. Lunghi, and M. Lavagna, "Tango Spacecraft Dataset for Region of Interest Estimation and Semantic Segmentation", Zenodo, Apr. 2022. DOI: 10.5281/zenodo.6507864.

[96] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[97] D. Wang, B. Wu, E. K. Poh, D. Wang, B. Wu, and E. K. Poh, "Dynamic models of satellite relative motion around an oblate earth", *Satellite Formation Flying: Relative Dynamics, Formation Design, Fuel Optimal Maneuvers and Formation Maintenance*, pp. 9–41, 2017.

[98] M Lavagna *et al.*, "Filtering techniques assessment towards pose estimation enhancement for image-based proximity navigation with uncooperative space objects", in *Aerospace Europe Conference 2023-Joint 10th EUCASS-9th CEAS Conference*, 2023, pp. 1–14.

[99] M. Piccinin, S. Silvestrini, G. Zanotti, A. Brandonisio, P. Lunghi, and M. Lavagna, "Argos: Calibrated facility for image based relative navigation technologies on ground verification and testing", 72^{nd} International Astronautical Congress, Dubai, United Arab Emirates, Oct. 2021.

[100] B. Razgus, E. Mooij, and D. Choukroun, "Relative navigation in asteroid missions using dual quaternion filtering", *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 9, pp. 2151–2166, 2017. DOI: 10.2514/1.G002805.

[101] J. L. Crassidis, F. L. Markley, and Y. Cheng, "Survey of nonlinear attitude estimation methods", *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 1, pp. 12–28, 2007. DOI: 10.2514/1.22452.

[102] G. Salemme, R. Armellin, and P. D. Lizia, "Continuous-thrust collision avoidance manoeuvres optimization", in *AIAA Scitech 2020 Forum*. 2020. DOI: 10.2514/6.2020-0231.

[103] J. L. Gonzalo, C. Colombo, and P. Di Lizia, "Analytical framework for space debris collision avoidance maneuver design", *Journal of Guidance, Control, and Dynamics*, vol. 44, no. 3, pp. 469–487, 2021.

[104] C. Bombardelli, "Analytical formulation of impulsive collision avoidance dynamics", *Celestial Mechanics and Dynamical Astronomy*, vol. 118, pp. 99–114, 2014.

[105] L. Gremyachikh *et al.*, "Space navigator: A tool for the optimization of collision avoidance maneuvers", *arXiv preprint arXiv:1902.02095*, 2019.

[106] F. K. Chan, *Spacecraft collision probability*. American Institute of Aeronautics and Astronautics, Inc., 2008.

[107] M. Vasile, V. Rodríguez-Fernández, R. Serra, D. Camacho, and A. Riccardi, "Artificial intelligence in support to space traffic management", 2018.

[108] R. Abay, M. Brown, R. Boyce, and A. Karacor, "Open source collision avoidance maneuver planning tool", Sep. 2017.

[109] N. Testa, *Reinforcement learning for the autonomous planning of debris collision avoidance strategies in densely populated environments*, 2021.

[110] A. R. Klumpp, "Apollo lunar descent guidance", *Automatica*, vol. 10, no. 2, pp. 133–146, 1974.

[111] C. D'Souza and C. D'Souza, "An optimal guidance law for planetary landing", in *Guidance, Navigation, and Control Conference*. AIAA, 1997. DOI: 10.2514/6.1997-3709. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.1997-3709.

[112] B. Acikmese and S. R. Ploen, "Convex programming approach to powered descent guidance for mars landing", *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007. DOI: 10.2514/1.27553.

[113] M. Szmuk and B. Acikmese, "Successive convexification for 6-DoF mars rocket powered landing with free-final-time", in *2018 AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, 2018. DOI: 10.2514/6.2018-0617.

[114] J. Guadagnini, M. Lavagna, and P. Rosa, "Model predictive control for reusable space launcher guidance improvement", *Acta Astronautica*, vol. 193, pp. 767–778, 2022, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2021.10.014.

[115] B. Gaudet and R. Furfaro, "Adaptive pinpoint and fuel efficient mars landing using reinforcement learning", *IEEE/CAA Journal of Automatica Sinica*, vol. 1, no. 4, pp. 397–411, 2014, ISSN: 2329-9274. DOI: 10.1109/JAS.2014.7004667.

[116] R. Furfaro, A. Scorsoglio, R. Linares, and M. Massari, "Adaptive generalized zem-zev feedback guidance for planetary landing via a deep reinforcement learning approach", *Acta Astronautica*, vol. 171, pp. 156–171, 2020, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2020.02.051.

[117] D. Iafrate, *Assessment of deep reinforcement learning for flexibility enhancement of planetary landing guidance and control*, 2023.

[118] B. M. Moghaddam and R. Chhabra, "On the guidance, navigation and control of in-orbit space robotic missions: A survey and prospective

vision", *Acta Astronautica*, vol. 184, pp. 70–100, 2021, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2021.03.029.

[119] S. Lee, G. Bekey, and A. Bejczy, "Computer control of space-borne teleoperators with sensory feedback", in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 205–214. DOI: 10.1109/ROBOT.1985.1087390.

[120] Z. Vafa and S. Dubowsky, "On the dynamics of manipulators in space using the virtual manipulator approach", in *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, 1987, pp. 579–585. DOI: 10.1109/ROBOT.1987.1088032.

[121] E. Papadopoulos and S. Dubowsky, "On the nature of control algorithms for free-floating space manipulators", *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 750–758, 1991. DOI: 10.1109/70.105384.

[122] F. Aghili, "Optimal control of a space manipulator for detumbling of a target satellite", in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 3019–3024. DOI: 10.1109/ROBOT.2009.5152235.

[123] T. Rybus, M. Wojtunik, and F. L. Basmadji, "Optimal collision-free path planning of a free-floating space robot using spline-based trajectories", *Acta Astronautica*, vol. 190, pp. 395–408, 2022, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2021.10.012.

[124] Y. Li, D. Li, W. Zhu, J. Sun, X. Zhang, and S. Li, "Constrained Motion Planning of 7-DOF Space Manipulator via Deep Reinforcement Learning Combined with Artificial Potential Field", *Aerospace*, vol. 9, no. 3, Mar. 2022, ISSN: 22264310. DOI: 10.3390/aerospace9030163.

[125] Z. Huang, G. Chen, Y. Shen, R. Wang, C. Liu, and L. Zhang, "An Obstacle-Avoidance Motion Planning Method for Redundant Space Robot via Reinforcement Learning", *Actuators*, vol. 12, no. 2, Feb. 2023. DOI: 10.3390/act12020069.

[126] S. Wang, X. Zheng, Y. Cao, and T. Zhang, "A Multi-Target Trajectory Planning of a 6-DoF Free-Floating Space Robot via Reinforcement Learning", in *IEEE International Conference on Intelligent Robots and Systems*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 3724–3730, ISBN: 9781665417143.

[127] M. D'Ambrosio, *Deep reinforcement learning aided robotics for uncooperative space asset grasping and in-orbit servicing*, 2023.

[128] J. Virgili-Llop, J. Ii, and M. Romano, "Spacecraft robotics toolkit: An open-source simulator for spacecraft robotic arm dynamic modeling and control", Mar. 2016.

[129] P. Colmenarejo *et al.*, "Methods and outcomes of the COMRADE project - Design of robust Combined control for robotic spacecraft and manipulator in servicing missions", in *69th International Astronautical Congress (IAC)*, 2018, pp. 1–5.

Colophon

This thesis was typeset with LaTeX and BibTeX, using a typographical look-and-feel created by Andrea Brandonisio. The style was inspired by G. Zanotti, A. Capannolo, S. Silvestrini, A. Colagrossi, D.A. Dei Tos `PhD_Dis` and by J. Stevens, L. Fossati `phdthesis` styles.

Thesis cover designed by Daniele Barberi Spirito. Image credits: *"Farragut's Fleet passing the Forts below New Orleans"* by Mauritz de Haas.